Merlyn inspect.

do
print
acce
canc

16/11/84 – a respeito do Merlyn

O Merlyn é o projeto de um computador pessoal de grande capacidade mas de preço relativamente baixo desenvolvido em função da linguagem Smalltalk 80. Ele representa, assim, uma segunda fase na chamada revolução dos micros por que permite um substancial aumento na produtividade daqueles envolvidos na informática além de tornar esta tecnologia acessível (e útil) a um número bem maior de pessoas. Assumpção/...

16/11/84 – a respeito deste diário

Este livro tem multiplos funções: ① monitorar o progresso do projeto ② acumular dados uteis para o planejamento futuro ③ concentrar em um único lugar todas as informações a respeito do projeto como um arquivo central.

Este aqui é o diário do Merlyn e não das pessoas envolvidas no projeto. Quem quiser fazer anotações estritamente pessoais deve manter seu próprio diário.

As anotações deverão estar em ordem cronológica sem nenhuma relação com o assunto ou qualquer outra organização formal.

Não deve ser usado um estilo muito formal: a linguagem deve ser mantida tão simples quanto for possível.

Não se deve omitir nada nem escrever demais. Isto não é um relatório de física, por isso deixe a enrolação fora disto.

Humor, se inevitável, deve ser pertinente, inteligente, curto e ter graça. Isto também não é uma coletânia de piadas.

Finalmente, quanto à forma das anotações, esta já é o segundo exemplo. Toda anotação começa pela data e assunto. O texto que se segue começa debaixo do assunto para deixar a data em evidência. Cada anotação trata de um assunto apenas. Um novo assunto começa tudo de novo deixando uma linha em branco entre as anotações. Todas as anotações terminam com um rabisco qualquer que identifique o autor. Não pode ter anotação anônima!

Mais um lembrete: quase tudo que hoje é tão obvio que não precisa ser escrito terá sido esquecido daqui a três meses e, portanto, precisa ser escrito!! Assumpção

22/11/84 - pré-história do projeto

No número de agosto de 1981 da revista Byte apareceram diversos artigos descrevendo a linguagem Smalltalk 80. O editorial dizia que apesar de não haver na época nenhum microcomputador com esta linguagem certamente dentro de pouco tempo haveria muitos. Alguns meses depois a Xerox lançou o Star que parecia uma boa máquina para o Smalltalk apesar deste não estar disponível. Eu achava

que dentro de poucos meses a Xerox lançaria a linguagem para o Star e se iniciaria o comprimento da previsão do editorial da Byte. Mas o Star custava mais de $17 mil; muito caro para a maioria das pessoas. Calculei que um computador semelhante com o MC68000 sairia abaixo de $10 mil e resolvi lançar meu próprio computador Smalltalk tão logo tivesse os recursos para construir pelo menos um protótipo. Voltei a estudar a Byte de agosto de 1981 mas minha compreensão da linguagem continuava muito limitada. Apesar disto resolvi acreditar na afirmação dos autores destes artigos de que crianças e outros pessoas expostas pela primeira vez a computadores aprendiam Smalltalk mais facilmente do que linguagens convencionais.

Em agosto de 1982 eu não tinha dinheiro nem para comprar um 68000 ($200 na época) quanto mais para levar o projeto adiante! Por isso quando apareceu a oportunidade de trabalhar com o 68000 no L.S.I. junto com o Osvaldo logo aceitei imaginando que poderia aproveitar os sistemas de desenvolvimento deles para continuar meu projeto. Infelizmente descobri que o L.S.I. não dispunha de muito mais recursos do que eu.

No início de 1983 a Apple lançava o Lisa, quase idêntico em tudo ao meu projeto. Muitos dos projetistas do Lisa vieram do projeto Smalltalk da Xerox e entre as linguagens prometidas para o sistema figurava exatamente o Smalltalk. Neste ponto o projeto é abandonado (erradamente pois o Lisa como o Star não cumpriria sua promessa).

Em meados de 1983 o Fabio me convida para ajudar no projeto de um computador para crianças. Resolvemos

desenvolver um computador com a linguagem Logo que fosse bastante barato. O hardware do projeto Pegasus era convencional exceto na ligação dos periféricos que era feita através de um esquema tipo rede local usando a linha serial do 6870 com transmissão em corrente diferencial e protocolo de passagem de "tokens". O software era mais interessante. A sintaxe mais complexa e tratamento de erros mais complexo do Logo em relação ao Lisp dificulta bastante uma implementação recursiva, tão popular nesta segunda linguagem. Além disso seria difícil manipular as multiplos pilhos num sistema multiprogramável como o nosso seria. Por isso resolvi emprestar os contextos ligados do iAPX 432. Testei esta idéia numa implementação do Logo em Basic no Texas Instruments TI 99/4A. Esta máquina era muito limitada para permitir muitas experiências mas mostrou que a idéia básica era boa além de mostrar o motivos de várias características "estranhas" de implementações disponíveis comercialmente. Assumpção

26/11/84 - pré-história do projeto: parte II

No fim de agosto de 1983 parecia que o projeto do supermini que desenvolvíamos no L.S.I. saíria até o final do ano deixando o laboratório com capacidade computacional para o projeto de VLSI. Combinei com o Osvaldo que aproveitaríamos estas ferramentas que nós mesmos estávamos construindo para aprendermos o projeto de circuitos integrados produzindo um que fosse útil comercialmente. Resolvemos

fazer um circuito que permitisse a construção de computadores cuja linguagem de máquina fosse o Smalltalk. Raciocinavamos que mesmo que na época em que fosse completado o projeto o mercado estivesse cheio de máquinas Smalltalk os computadores construidos com nosso chip seriam bem mais rápidos.

Apesar de decidirmos que só iniciariamos este projeto quando todos os outros estivessem acabados eu voltei a estudar a Byte de agosto de 1981 para ver se era viável. Descobri que algumas idéias que eu estava usando no Logo poderiam ser aplicadas ao Smalltalk também. Resolvi investigar se o contrário também era verdadeiro. Inventei uma notação para incluir mensagens do tipo do Smalltalk no Logo e incorporei classe e objeto também. Descobri que assim podia dispensar as "propriedades" do Lisp e Logo e que a linguagem se tornaria muito mais uniforme e fácil de implementar. Para testar estas idéias resolvi escrever um grande programa neste novo Logo, por exemplo um interpretador para este Logo completo. (A esta altura o hardware já estava pronto a bastante tempo. Um assembler para o 6809 em Lisp no TRS 80 do Fabio não podia ser compilado por falta de memória de modo que estavamos escrevendo todos os programas em hexadecimal. Tinhamos testado as várias partes do hardware com "loops" bem simples e um osciloscópio emprestado. Problemas bem graves de ruído e interferência fizeram com que fosse difícil testar programas mais complexos sem nenhum equipamento de apoio. Passamos, então, a nos concentrar mais no software.) O interpretador saiu

melhor do que eu esperava e como os empresários que tinham tido a ideia original tinham desaparecido resolvi mudar a orientação do projeto e dar ênfase aos aspectos mais sofisticados do projeto dotando-o de uma interface com usuário semelhante ao Smalltalk.

Nesta época a capacidade computacional disponível para o projeto era apenas o TRS 80 do Fabio. O acesso a esta máquina não estava muito conveniente para mim e resolvi interromper o desenvolvimento propriamente dito e montar um Apple II para reforçar a infraestrutura do projeto. A esta altura dos acontecimentos (dezembro) já tinha traduzido o interpretador para assembler do 6809 mas só no papel.

Em março de 1984 iniciei a montagem do Apple mas até julho este ainda não estava completo por falta de dinheiro para comprar peças. Durante este tempo procurei reforçar a teoria por trás do projeto e pude adquirir bastante experiência com os fundamentos da programação por objetos.

Em julho de 1984 eu e o Fabio concordamos que o projeto não estava satisfazendo nenhum de nós. Estava muito caro e complicado para o que ele queria e muito limitado para o que eu queria. Resolvemos desmembra-lo em dois projetos: ele continuaria com o Pegasus, um computador Logo barato para crianças, e eu ficaria com o Merlyn, uma avançada máquina Smalltalk de uso geral. Assumpção/x.

21/1/85 - história do projeto

Em julho de 1984 recomecei o projeto. O Oswaldo logo resolveu me ajudar. Definimos o Merlyn mais ou menos como um Sinclair QL com Smalltalk com bastante do Macintosh misturado no meio. O hardware seria simples mas muito poderoso. O "coração" do sistema seria o "Raster Memory System" da Motorola que geraria todo o vídeo gráfico colorido além de controlar 128k de memória dinâmica para o 68008. O teclado e o "mouse" seriam controlados por um 68705 enquanto os outros periféricos todos seriam ligados ao computador por uma rede local controlada pelo WD2840. Todo o sistema (incluindo ROM suficiente para conter a linguagem) caberia numa caixa não muito maior do que o teclado e que seria ligado a uma televisão colorida comum.

O software desta máquina teve um grande avanço quando adquirimos "Smalltalk 80 - The Language and It's Implementation". Este livro esclareceu as muitas dúvidas a respeito do uso e funcionamento da linguagem além de listar todos os comandos desta.

O planejamento do desenvolvimento estava razoavelmente detalhado e era, de uma maneira bem resumida, assim: Implementaríamos o Smalltalk como descrito no livro no Apple e ao mesmo tempo construiríamos o hardware. Reescreveríamos, então, a "Máquina Virtual" para o 68008 passando a "Imagem Virtual" sem alterações. A partir daí dividiríamos nossos esforços entre o fim dos manuais (começado nas etapas anteriores) e desenvolvimento de periféricos e o estabelecimento das infraestruturas de fabricação e comercialização; com grupos externos fazendo os aplicativos.

Assumpção/n.

22/1/85 - história do projeto : parte II

Apesar desta ser a parte mais recente é a que está menos clara em minha mente. Por isso vou descreve-la em duas etapas: informações que colhemos de outros projetos, e o alteramos ~~se~~ nosso.

A primeira grande preocupação nossa foi com o projeto SOAR de Berkley. Eles já tinham construído dois rápidos e simples microprocessadores de 32 bits segundo a filosofia RISC e agora queriam fazer um otimizado para o Smalltalk. Enquanto aguardávamos a publicação dos detalhes deste projeto no STG Arch da ACM procuramos descobrir mais detalhes através de artigos anteriores e concluímos que eles deviam ter baseado seu esquema de gerenciamento de memória no "A Real-Time Garbage Collector Based on The Lifetimes of Objects" de Henry Lieberman e Carl Hewitt, o que veio a ser confirmado. Outros resultados deste projeto não foram tão interessantes.

A Tektronix foi a primeira firma a lançar um computador Smalltalk comercial. O Tektronix 4404 Artificial Intelligence System custa quinze mil dolares e é baseado no 68010 com 20 megabeytes de disco.

A Atari acaba de lançar a família ST de computadores baseados no 68000. A versão de 128KB custa $400 enquanto a de 512KB custa $600. O sistema vem com Basic e Logo em Rom e se parece muito com a descrição que fizemos do Merlyn em julho de 1984.

Na segunda parte, ao tratar do nosso trabalho neste período, não vou seguir a ordem cronológica mas sim a ordem em que vou me lembrando dos eventos.

No início, nossa implementação do Smalltalk seria a apresentada no livro mencionado anteriormente, mas já pensávamos em implementações bem diferentes para máquinas futuras. Depois de algum tempo, a medida que as idéias para esta implementação alternativa foram ficando mais definidas, resolvemos pulsar após fazermos funcionar a versão do Smalltalk da Xerox no Apple tentaríamos a nossa própria versão na mesma máquina, passando a melhor das duas para o protótipo. Hoje estamos considerando implementar já de cara a nossa versão, abandonando por completo a da Xerox.

Logo no início nos pareceu que o 68008 teria uma economia muito pequena em relação ao 68000, razão pela qual decidimos substituí-lo por este último para aumentar o desempenho do nosso computador.

Várias idéias foram consideradas mas abandonadas tais como o uso de memórias bolha para armazenar o sistema no lugar da ROM, circuitos para aumentar a velocidade de certas operações como o "bit block transfer" para gráficos, além de outras idéias do mesmo estilo. Abraços p.

23/1/85 - história do projeto: parte III

Uma das partes que mais sofreu alterações ao decorrer do projeto é a ligação com os periféricos. A idéia básica sempre foi um esquema tipo rede local que pudesse interligar vários

computadores entre si e onde os periféricos fossem "servos" desta rede. A primeira idéia tinha sido usar o 2840 da Western Digital. Me pareceu, no entanto, que esta escolha não era muito boa do ponto de vista comercial pois é a única parte que teria que permanecer constante em todos os projetos futuros, o que significa que teríamos eternamente problemas de fornecimento e importação pois esta peça não tem outros fornecedores. Depois de um pouco de pesquisa pareceu-me uma alternativa interessante uma rede em anel com passagem de "tokens" feita apenas com TTL's. Tal rede seria completamente assíncrona pois o receptor aproveitava a capacidade de transmissão bidirecional da linha para confirmar o recebimento de cada bit. Esta rede apresentava vários problemas teóricos e práticos de modo que consumiria a maior parte do tempo de desenvolvimento do Merlyn.

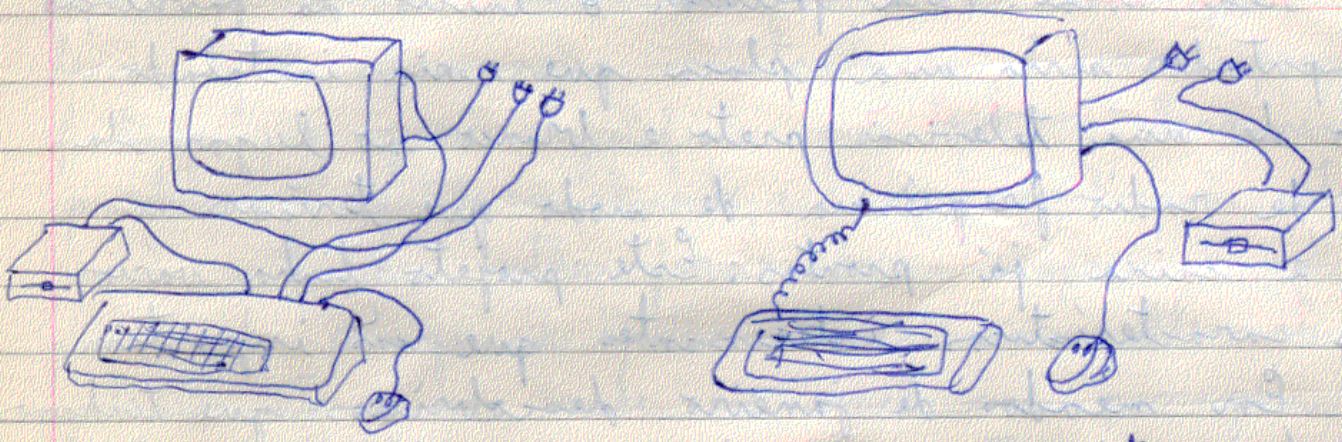
Para melhor compreender as próximas alterações que fizemos no projeto vale a pena mencionar outro trabalho que fizemos na segunda metade de dezembro e na primeira metade de janeiro. O José Carlos (Zeca) e o Fabio ficaram sabendo que a USP abriria concorrência para a compra de 2000 microcomputadores e acharam que seria uma boa oportunidade para entrar nesta área. Eu e o Osvaldo decidimos que isto poderia capitalizar o nosso projeto do Merlyn e resolvemos participar deste outro. Este computador seria uma máquina CP/M, mas achamos que deveria ter portas

do tipo IEEE 488 (ou GPIB) para poder ser ligado aos instrumentos que muitos laboratórios desta escola já possuem. Além disto vi que com dois portos GPIB os computadores poderiam ser ligados entre si como numa rede local. Para facilitar o início da fabricação o computador seria uma placa que seria instalado dentro de uma televisão preto e branco no lugar da parte de radio frequência de modo que teríamos a fonte e caixa já prontos. Este projeto tinha vários outros características interessantes que não importam aqui. Em meados de janeiro descobrimos que tínhamos sido informados erroneamente a respeito de datas e prazos da concorrencia e que além de estar muito em cima da hora para entrar nela, ela havia sido arranjada para favorecer certo fabricante.

Enquanto os outros ainda não se definiram sobre o que querem fazer com este projeto eu e o Osvaldo voltamos a nos concentrarmos no Merlyn que estava a mais de um mes parado. Resolvemos aproveitar algumas ideias desta máquina CP/M no Merlyn. Trocamos nossas duas portas seriais assíncronas pelos duas portas GPIB que apesar de mais caras já estão completamente definidas e são compatíveis com uma vasta quantidade de periféricos já existentes.

A outra alteração veio do fato que já havíamos constatado de que mesmo quando o computador pode ser ligado a uma televisão comum aqui no Brasil o usuário quase sempre acaba comprando uma televisão ou monitor só para o computador. Foz, portanto, mais sentido já incluir o monitor como parte do sistema

básico. Resolvemos transferir a placa principal da caixa do teclado para a caixa do monitor como no computador CP/M. O sistema fica, assim, muito mais conveniente, como mostra a figura:



CPU no teclado

CPU no monitor

Decidimos que apesar de que o circuito da CPU teria de qualquer maneira a capacidade de gerar cores (característica do RMS) o monitor do primeiro modelo seria preto e branco. Depois de iniciada a venda deste poderíamos fazer um modelo "profissional" a cores. ~~Assunção P.~~

08/02/85 - formato de disquete.

A idéia surgida para formatação dos disquetes é a seguinte: a unidade de informação é a trilha, ao invés do setor — a razão está no fato que é possível armazenar em buffers próprios da unidade de disco o conteúdo de toda uma trilha a um custo (hoje) pequeno.

O tamanho (lógico) dos setores torna-se irrelevante e o que é melhor, é possível (sem maiores complicações) guardar uma quantidade de informações proporcional ao raio da trilha

corrente.

Para marcar o início da trilha (início lógico, naturalmente) temos um sinal especial, decorrente do fato de utilizarmos um formato de gravação parecido com o do Apple - RLL. Cristo.

25/02/85 - movimento financeiro

Recebi de Leda Cecília Assumpção a quantia de Cr$ 295 mil correspondentes a mais ou menos US$ 80,36 para a encomenda por via aérea dos seguintes livros: 'Smalltalk-80: the interactive programming environment' e 'Smalltalk-80: bits of history, words of advice'.

Nesta fase inicial do projeto anotaremos aqui todo gasto e todo lucro, sempre sob o título de 'movimento financeiro'. Ao contrário desta anotação que se refere a um movimento de duas semanas atrás, todas as outras anotações deverão ser feitas no próprio dia em que for feita a transação a que se refere. Assumpção Jr.

26/2/85 - Smalltalk em Smalltalk

O programa abaixo descreve a parte do sistema Smalltalk que será escrita eventualmente em linguagem de máquina e que interpreta os métodos compilados. É interessante notar a semelhança entre as estruturas estáticas (carimbos e blocos) e as estruturas dinâmicas usadas pelo interpretador. Estas estruturas ocupam mais memória do que os 'byte-codes' usados por

outros sistemas Smalltalk mas permitem uma uniformidade muito maior, o que esperamos resulte numa velocidade de execução muito maior.

Só serão mostrados os métodos que esclareçam o funcionamento do interpretador:

```
>> classe: Carimbo 'expressões' ~~'é Constante'~~
>> superclasse: Objeto
"unidade básica do código objeto Smalltalk. representa uma única mensagem com a Coleção Indexada expressões contendo o seletor, o receptor e os parametros da mensagem, ~~enquanto que a Coleção Indexada é Constante indica quais expressões correspondentes são constantes~~"
remetente: contexto1 índice: umInteiro maxLex: contexto2
    |temp| temp ← ContextoDeMétodo criar.
        temp remetente: contexto1 índice: umInteiro
            ~~maxLex: contexto1 maxLex~~ maxLex: contexto2.
        temp expressões: expressões copiaRápido.
        ~~é Constante: é Constante copiaRápido.~~
    ↑temp.

guardarContexto
    || ↑ auto remetente: ProcessoAtual contextoAtual
            índice: ProcessoAtual índice maxLex:
            ProcessoAtual maxLex

avaliar
    || ProcessoAtual dispachar: auto guardarContexto.

>> classe: CarimboDeResposta
>> superclasse: Carimbo
```

remetente: contexto1 índice: umInteiro maxLex: contexto2
    || ↑ super remetente: contexto2 remetente índice:
        contexto2 índice maxLex: contexto2

>> classe: SuperCarimbo
>> superclasse: Carimbo.
remetente: contexto1 índice: umInteiro maxLex: contexto2
    |temp| temp ← SuperContexto criar.
        temp remetente: contexto1 índice: umInteiro
            maxLex: contexto2.
        temp expressões: expressões copiarRápido.
        ↑temp

>> classe: ~~Carimbo~~ Literal
>> superclasse: Carimbo
avaliar
    || ProcessoAtual responder: expressões para:
        ProcessoAtual contextoAtual índice:
        ProcessoAtual índice maxLex: ProcessoAtual
        maxLex

>> classe: RespostaLiteral
>> superclasse: Carimbo
avaliar
    || ProcessoAtual responder: expressões para:
        ProcessoAtual maxLex remetente índice:
        ProcessoAtual maxLex índice maxLex:
        ProcessoAtual maxLex maxLex

>> classe: Bloco 'temporários'
>> superclasse: Carimbo

remetente: contexto1 índice: umInteiro maxLex: contexto2
    |temp| temp ← ContextoDeBloco criar.
        temp proxLex: contexto1.
        temp expressões: expressões. "não
            precisa ser uma cópia pois os
            valores devolvidos não são guardados"
        temp temporarios: (ColeçãoOrdenada criar:
                temporarios).
        ↑ temp

avaliar
    | | ProcessoAtual responder: auto guardarContexto
        para: ProcessoAtual contextoAtual índice:
        ProcessoAtual índice maxLex: ProcessoAtual
        maxLex

>> classe: Contexto 'remetente índice maxLex expressões'
>> superclasse: Objeto
remetente: contexto1 índice: umInteiro maxLex: contexto2
    | | remetente ← contexto1.
        índice ← umInteiro.
        maxLex ← contexto2

expressões: umaColeçãoOrdenada
    | | expressões ← umaColeçãoOrdenada

em: índice por: valor
    | | expressões em: índice por: valor

nívelLéxico: umInteiro
    | | ↑ remetente nívelLéxico: umInteiro

```
>> classe: ContextoDeBloco 'temporários proxLex'
>> superclasse: Contexto
temporários: uma ColeçãoOrdenada
    || temporários ← uma ColeçãoOrdenada

proxLex: umContexto
    || proxLex ← umContexto

valor
    || remetente ← ProcessoAtual ContextoAtual.
       índice ← ProcessoAtual índice.
       maxLex ← ProcessoAtual maxLex.
       ProcessoAtual dispachar: auto

valor: umValor
    || temporarios em: 1 por: umValor.
       auto valor

avaliarEm: umInteiro
    || (umInteiro = expressões tamanho)
          seVerdade: [ProcessoAtual voltar]. "responda
                para quem eu respondo"
       (expressões em: umInteiro) avaliar

temporário: umInteiro
    || ↑ temporários em: umInteiro

temporário: umInteiro por: valor
    || temporários em: umInteiro por: valor
```

```
em: índice por: valor
    "não faça nada; jogue fora o valor calculado"

nívelLéxico: umInteiro
    (umInteiro = 1) seVerdade: [ ↑ avto ]
                    seFalso: [ ↑ proxLex nívelLéxico:
                                 umInteiro - 1 ]

>> classe: ContextoDeMétodo
>> superclasse: Contexto
avaliarEm: umInteiro
    (umInteiro > expressões tamanho)
        seVerdade: [ ProcessoAtual enviar ]
        seFalso: [ (expressões em: Inteiro) avaliar ]

classeDoReceptor
    ↑ avto receptor classe

receptor
    ↑ expressões em: 2

seletor
    ↑ expressões em: 1

parametro: umInteiro
    ↑ expressões em: 2 + umInteiro

>> classe: SuperContexto
>> superclasse: ContextoDeMetodo
classeDoReceptor
    ↑ ~~superclasseDoReceptor superclasse~~
```

↑ (maxLex receptor classe que Classe Inclui Seletor:
maxLex seletor) superclasse

>> classe: Interpretador 'contextoAtual índice maxLex'
>> superclasse: Objeto
contextoAtual
    || ↑ contextoAtual

índice
    || ↑ índice

maxLex
    || ↑ maxLex

avançar
    || índice ← índice + 1.
    contextoAtual avaliarEm: índice

dispachar: novoContexto
    || contextoAtual ← novoContexto.
    índice ← 0.
    auto avançar

voltar
    || índice ← contextoAtual índice.
    maxLex ← contextoAtual maxLex.
    contextoAtual ← contextoAtual remetente

responder: valor para: contexto1 índice: umInteiro
maxLex: contexto2
    || índice ← umInteiro.

```
        maxLex ← contexto2 .
        contextoAtual ← contexto1 .
        contextoAtual em: índice por: valor.
        auto avançar

enviar
    | método valor |
    método ← auto procurar: contextoAtual seletor
                em: contextoAtual classeDoReceptor.
    método éNulo seVerdade: [ auto
                enviarSeletorNãoEntendido ]
    valor ← PrimitivoFalhou.
    método primitivo éNulo seFalso: [ valor ← auto
                fazerPrimitivo: método primitivo ]
    (valor = PrimitivoFalhou)
        seFalso: [ auto responder: valor para:
                contextoAtual remetente índice:
                contextoAtual índice maxLex:
                contextoAtual maxLex ]
        seVerdade: [ maxLex ← contextoAtual. valor
                método compilado avaliar ]


procurar: seletor em: umaClasse
    | valor | Cache contém: seletor e: umaClasse
            seVerdade: [valor ← Cache em: seletor
                e: umaClasse ]
            seFalso: [ valor ← umaClasse procurar:
                seletor.
                valor éNulo seFalso: [
                Cache em: seletor e: umaClasse
                por: valor . ]]
    ↑ valor
```

Para escrever este trecho acabei desistindo de rabiscar este caderno e desenvolvi-o num processador de texto onde podia ir para frente e para trás no programa fazendo as alterações necessárias. Neste processo esqueci dos comentários de modo que a listagem que copiei aqui está mais "pelada" além de incompleta. #Bumpzinho/h:

30/3/85 - Lançamento

A firma Digitalk anunciou no número de janeiro de 1985 da revista Byte o sistema "Methods" para o IBM PC e compatíveis por 250 dolares dizendo ser compatível com o Smalltalk-80. O fato do sistema precisar de 512kB não é surpreendente mas o nome do programa é: isto provavelmente significa que a Xerox reserva o nome Smalltalk-80 para a sua implementação e que nós talvez tenhamos que adotar outro nome para a nossa versão. #Bumpzinho/h:

30/3/85 - Livros

Já faz uma semana que recebi os livros "Smalltalk-80: Bits of History, Words of Advice" e "Smalltalk-80: the ~~Interactive Graphical User Interface~~ Interactive Programming Environment" ambos fundamentais para o prosseguimento do projeto. O sistema excede minhas espectativas e mostra claramente ser fruto de dez anos de pesquisa e trabalho. Em face disto, nossos planos de produzir certos avanços teóricos além de reimplementar tanto em tão pouco tempo parecem um tanto exagerados. #Bumpzinho/h:

10/4/85 - Projeto do Hardware

Como já indiquei anteriormente, o projeto do Hardware vem sofrendo muitas alterações, praticamente todas na parte da rede local. Aliás, esta parte tem que ser feita com muito cuidado pois é o que deve ser mantido constante em projetos futuros. Não deve nem ser muito caro hoje, nem muito limitado amanhã. A rede com portas IEEE 488 seria um tanto cara hoje, mas por ser assíncrona prometia capacidade de expansão. O problema é que a topologia em anel com "store-and-forward" de pacotes só funciona com todas as estações ligadas e em ordem. A solução seria que as duas portas fossem curto circuitadas quando a estação não estivesse ativa, mas devido ao grande número de linhas isto não é muito prático. Na área de topologia em "bus" existem muitos movimentos mas poucos padrões. O Ethernet é muito caro e continuará a sê-lo por muito tempo ainda. O Omninet parece mais interessante mas ainda é um pouco caro ($400-500 por estação) e seu uso exigiria um licenciamento da Corvus. Uma vantagem desta última rede local é que ela não usa circuitos "exóticos" mas sim o padrão RS 422 em "twisted pair" para transmissão e recepção e o MC6854 Advanced Data Link Controller para cuidar do protocolo de baixo nível. Desconheço o método de conecção mecânica adotado e não está muito claro como são os protocolo de mais alto nível.

A ideia no Merlyn é fazer algo parecido

já que não dá para ser totalmente compatível.

A interface com a rede local usa um canal de DMA. Pensamos em usar um MC68450 ou MC68440 para esta função mas um MC6844 é mais barato e fácil de achar e neste projeto será possível contornar algumas de suas limitações com um pouco de imaginação.

Toda outra entrada e saída pode ser controlada por um microcomputador em um só chip tal como o MC68705P5. Ele leria do teclado e do "mouse" e geraria o som. Estas interfaces usariam mais software do que hardware. A comunicação com o "host" seria através de outro canal de DMA de modo que o uso de "buffers" é controlado ~~pelo~~ indiretamente pelo processador principal. Uma vantagem adicional é que fica mais difícil copiar pois não há como o usuário ler o programa que foi gravado. Thompson

10/4/85 - Disponibilidade de Componentes

A Motorola acaba de me avisar que o MC68486 e MC68487 não estão em produção ainda. Por enquanto só informações preliminares sob "non disclosure agreement", amostras só no fim deste semestre e quantidade no 3º trimestre.

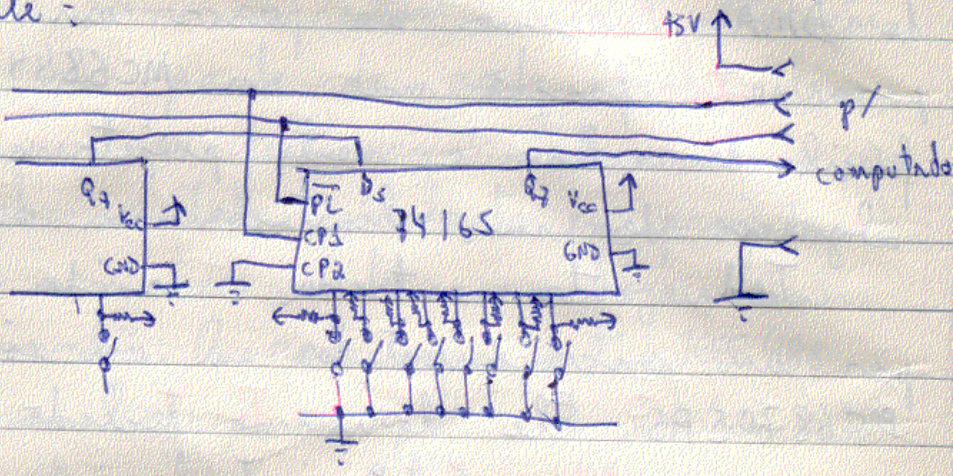
Temos que rever todos nossos planos. Thompson

12/4/85 - Idéias para o Teclado

Para o Smalltalk é melhor um teclado não decodificado, ou seja: ao invés de receber o código ASCII correspondente às teclas pressionadas o computador lê diretamente o estado

de cada tecla e faz o resto por software. Uma maneira barata de fazer um teclado não decodificado serial é a seguinte:

8 x 74165 =



Ao ~~transmitir um nível~~ ~~provoco uma transição~~ ~~de 0 para~~ Ø na linha $\overline{P_L}$ o computador armazena o valor de todas as 64 teclas no "shift register" de 64 bits formados pelos 8 74165. O ~~valor~~ estado da primeira tecla aparecerá na linha Q7. Toda vez que ocorrer uma transição de 0 para 1 na linha CP1 o estado da próxima tecla aparecerá em Q7. ~~Assumindo~~

17/4/85 - Alternativas de Hardware

Tendo visto a demora prevista pela Motorola no fornecimento do "Raster Memory System" temos as seguintes alternativas no desenvolvimento do Hardware: ① esperamos o RMS da Motorola ② usamos algo parecido de outro fabricante como o ACRTC da Hitachi ③ usamos algo mais antigo como o 6845 da Motorola ④ fazemos o vídeo discreto como no Macintosh ⑤ fazemos como na opção anterior mas substituímos o 68000 por uma implementação em hardware do Smalltalk.

A alternativa 1 não é muito promissora já

que os prazos dados pela Motorola são exatamente os mesmos dados a um ano atrás; ou seja: não devemos contar com eles.

A 2 também tem seus problemas. Estes sistemas são inspirados em padrões gráficos como o GKS e dificultam demais a implementação do Bit Block Transfer.

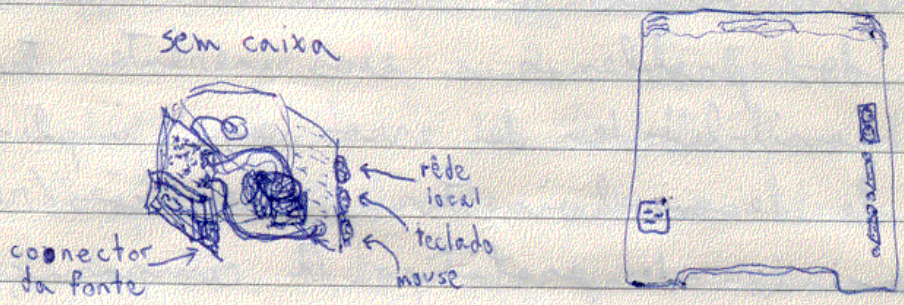
O principal problema com o 3 é que estes chips exigem muitos TTL's de suporte de modo que pode ficar menos econômico que a alternativa 4 se esta for bem feita.

Desta forma as alternativas 4 e 5 são as melhores apesar dos problemas a elas inerentes tais como a grande possibilidades de erros que implicam um grande tempo de desenvolvimento e alterações/reconstruções substanciais do protótipo. A comparação entre a 4 e a 5 é bem mais complicada. O desenvolvimento da 4 é bem mais simples sendo um microcomputador enquanto o desenvolvimento da 5 é no estilo dos miniconputadores. Para avaliar as duas alternativas em termos de custo/desempenho creio que só depois de implementarmos o Smalltalk em software e em seguida detalharmos completamente o hardware das duas alternativas. Em relação ao desempenho o interpretador deve ser mais rápido em hardware enquanto no 68000 muito mais funções podem ser primitivos. ~~Assunção~~/1.

18/4/85 - Ideias para a montagem mecânica

O ideal seria que todo o sistema cabesse numa única placa. Na prática provavelmente

teremos que usar duas placas. Assim sendo é melhor dividirmos o sistema em duas partes: placa digital (o computador) e placa analógica (fonte e monitor). As duas placas podem ficar em pé (para dissipação de calor) uma de cada lado do tubo de imagem. Como há poucas ligações entre as duas, estas podem ser feitas por um feixe de fios. Os conectores com o mundo externo devem estar dispostos de tal forma que possam ser soldados diretamente nas placas. Uma possível montagem seria (vista de trás):



Neste desenho indiquei a ligação com a rede como dois conectores ~~DIN~~ de 3 pinos à la "appletalk" (aliás tem um excelente artigo a respeito da rêde da Apple no número de Março de 1985 da revista A+) enquanto o mouse e o teclado usariam conectores do tipo "subminiture D" ou joystick Atari de 5 pinos. Seria melhor que estes dois ultimos fossem diferentes para que não pudessem ser confundidos. ~~A~~ Bruno P.

18/4/85 - Ideias para o Video

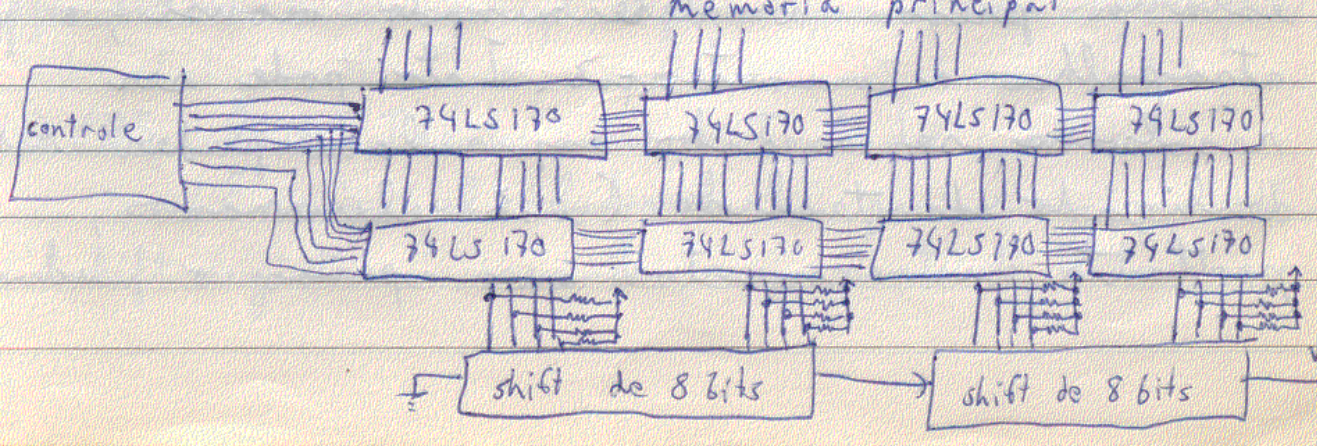
Num tubo de 12 polegadas deixando uma borda razoável e com uma resolução de 640 x 480 teremos 74 pontos por polegada nos dois eixos produzindo uma imagem comparável a uma impressora de matriz de pontos e sem distorções (alongamento ou achatamento

de figuras).

Seria interessante montar o tubo de lado de modo que fosse mais alto do que largo e, portanto, mais parecido com folhas de papel a que as pessoas já estão habituadas. Outra vantagem desta posição do tubo é que a montagem descrita na página anterior acomodaria placas maiores.

Em quase todos os projetos de controlador de vídeo, este consome quase metade do tempo da memória. A situação apresenta várias complicações adicionais quando o "host" é um microprocessador assíncrono como o 68000. Uma memória "FIFO" entre a memória principal e os "shift registers" do vídeo simplifica o circuito ao dar uma tolerância razoável no atraso entre o vídeo pedir a memória e conseguí-la. Se o projeto aproveitar modos rápidos de acesso às memórias dinâmicas tais como o "nibble mode" e o "page mode" para o vídeo (o que não pode ser feito sem o FIFO) então será muito aumentada a porcentagem do tempo que a memória fica disponível para o microprocessador.

O problema é achar uma FIFO comum e barata. Um quebra galho meio caro é usar oito 74LS170 4x4 file registers como um buffer circular 8x32 bits:



Na inicialização do sistema o controle enche os dois bancos do "FIFO" e "solta" o vídeo que lê uma nova palavra toda vez que o shift register se esvazia, percorrendo circularmente os 8 endereços. Toda vez que o vídeo lê a última palavra de um banco o controle pede a memória principal e quando consegue preenche completamente este banco usando o "nibble mode". Szwarcfiter.

23/4/85 - Idéias para os Conectores

Num item anterior, a respeito da montagem mecânica, estava pensando em cinco conectores: um para a alimentação de alta tensão, dois para a rêde, um para o mouse e um para o teclado. O da alimentação é bem distinto dos outros. Os dois da rede são iguais entre si e diferente dos outros, o que não causa nenhum problema já que são intercambiáveis. O que poderia causar alguma confusão é o fato do conector do mouse ser igual ao do teclado. Acho que a melhor solução seria usar um só conector para os dois. Isto também descongestionaria atrás do Merlyn. Um inconveniente é que determinado teclado estaria "preso" a determinado mouse, o que atrapalha a manutenção. Isto pode ser resolvido se o cabo espiralado puder ser desligado do teclado (não eliminamos, portanto, um conector mas apenas o mudamos

de lugar). Assumpção Jr.

15/12/85 - Resumo das Atividades

Neste grande intervalo sem nenhuma anotação houve muita atividade mas pouco progresso. Neste período eu trabalhei Tempo integral na parte de Sistema Operacional na Softec com o Ruy. O objetivo deste trabalho foi o de conseguir dinheiro para componentes e outros aspectos do projeto, mas como efeito colateral sobrou pouco tempo para trabalhar no Merlyn em si. Desta forma foi possível comprar o programa Methods (Smalltalk para o IBM PC) e os componentes para montar um primeiro protótipo.

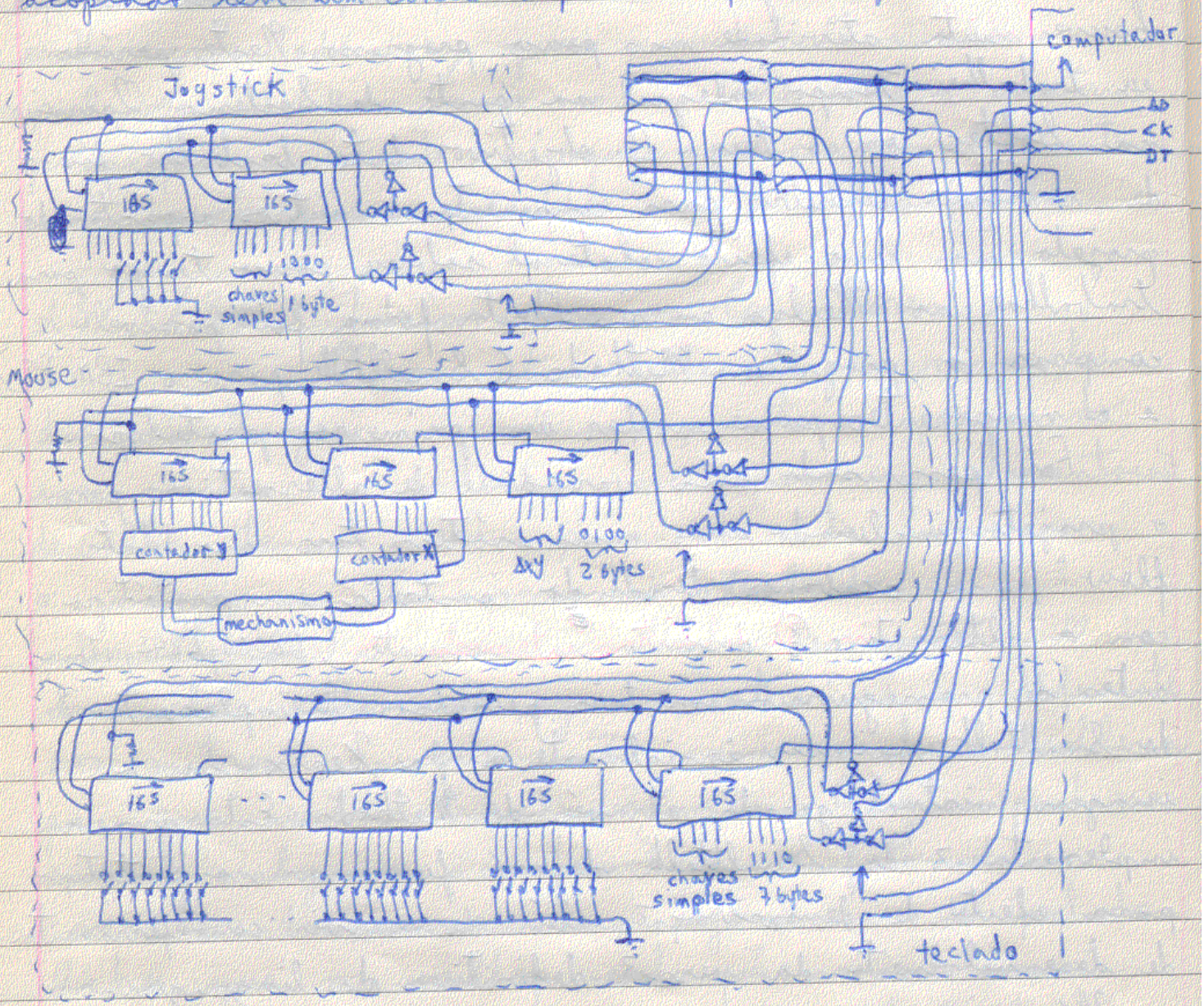
Foi pesquisada uma variedade de alternativas para o projeto, incluindo umas mais exóticas tais como "data flow", mas acabamos decidindo construir um protótipo com a alternativa ③ descrita anteriormente (68000, 6845, etc...) antes de prosseguirmos com tais pesquisas. A implementação do Smalltalk vai ser inicialmente a da Xerox com a imagem virtual gerada através do Methods. Esta primeira implementação do Smalltalk deverá funcionar no protótipo para efeito de demonstração além de servir como ferramenta de desenvolvimento da versão definitiva da linguagem.

Há mais ou menos um mês atrás eu e o Osvaldo convidamos o Fabio para trabalhar conosco cuidando dos programas aplicativos. Ele aceitou mas falou que gostaria de fazer também a documentação, o que nos parecem bom. Assumpção Jr.

21/12/85 - Mais Idéias para o Teclado

É possível elaborar um pouco mais a idéia dos "shift registers" criando uma porta de entrada

mais genérica. O primeiro byte "shiftado" pelo computador indicaria o número de bytes de informação que seguem e como interpretar estes bytes. Cada dispositivo de entrada acoplado tem um conector para acoplar o próximo:



Nesta figura vemos um joystick, um mouse e um teclado alfanumérico de 56 teclas ligados na porta de entrada. O computador lê a descrição do dispositivo seguida dos bytes de informação. O ultimo destes bytes é imediatamente seguido pela descrição do próximo dispositivo, etc... O "pull down" na entrada do primeiro "shift register" garante que zeros serão lidos depois

do último periférico. Um periférico cuja descrição indica zero bytes de informação não existe e indica que tudo que estava ligado à porta de entrada já foi lido. A ligação física consiste em um par de conectores de cinco pinos, um macho e uma fêmea, acoplados mecânicamente numa única peça e interligados ao aparelho por um cabo de oito vias, espiralado ou não. A função única dos inversores é reamplificar os sinais recebidos. O atraso introduzido por estes inversores não é significativo e não tem problema de acumular com dispositivo sucessivos. Note que os contadores do mouse são gerados toda vez que são lidos, de modo que esta leitura não deve ser abortada e mais tarde reiniciada. ~~Assumpção~~ Fi

21/4/86 - Mais um Resumo das Atividades

Na época da última anotação o Osvaldo passou a limpo o esquema para o primeiro protótipo. Acabamos fazendo algumas alterações, de maneira que os componentes já comprados não correspondiam exatamente aos que usaríamos.

Logo depois disto, falei para o Rui e para o Hélio (dono da Softec) que iria passar a trabalhar tempo integral no Merlin. Ao ver o esquema do projeto e uma demonstração do Methods, o Hélio mostrou-se interessado em participar do projeto. No fim de dezembro e no início de janeiro eu e o Osvaldo tivemos várias reuniões com o Hélio e começamos a definir como meio mais viável de colaboração o formarmos uma nova empresa para desenvolvermos o projeto que seria então licenciado para a Softec fabricar e comercializar. Em meados de janeiro o Fabio e o Osvaldo resolveram que não poderiam largar outros projetos nos quais estavam

envolvidos para dar uma maior dedicação ao Merlin. Resolvi, então, continuar minhas negociações com o Hélio sozinho. No fim de fevereiro foi marcado em minha carteira de trabalho a minha saída da Softec. A esta altura já tínhamos resolvido que o Hélio teria a maioria desta nova firma, o Jecel (meu pai) e eu teríamos o resto. O Ruy resolveu que não teria nenhuma participação. Semana passada o Hélio sugeriu que seria mais interessante mudar a razão social de uma firma pré-existente (acho que o nome da firma em que ele está pensando é Marketing Analises) do que criar uma nova.

Enquanto isto, em dezembro eu comecei a traduzir para a linguagem "C" o interpretador de Smalltalk como descrito no livro "Smalltalk 80: the language and it's implementation". Até o fim de fevereiro eu já tinha os arquivos interpreter.c, bytecodes.c e primitives.c no Apple e os transferimos para o Ego. A partir desta data eu voltei a trabalhar no prédio da Softec e escrevi os arquivos objectMemory.c e st.c no Ego. Eu gastei a última semana e março e a primeira de abril tentando fazer com que este programa carregasse a "imagem" criada pelo Methods para eliminar todos os erros do objectMemory.c.

27/4/86 - Construção do Hardware

Em meados de março o Roberto, que trabalha como estagiário na Softec, passou a limpo algumas alterações que eu havia feito no projeto e iniciou a construção do primeiro protótipo no conjunto de placas

que meu pai e o Osvaldo construiram em dezembro. Sob a orientação do Ruy, todos os conectores do Merlin foram feitos compatíveis com o Ego para podermos aproveitarmos fontes e monitores do laboratório. No início de abril o protótipo já estava pronto e parcialmente testado, faltando apenas o processador.

No dia 11 de abril a Phillip forneceu uma amostra do 68000 e passamos então à depuração do protótipo. Como eu ainda não havia conseguido instalar o Asembler no Ego, escrevi um programa a mão que inicializaria o 6845 e entraria em loop. Como o Merlin não funcionou e o "pseudo-analisador lógico" já estava trancado em um armário (era depois do expediente), nós deixamos para depura-lo na segunda-feira.

Infelizmente, segunda perdemos o dia inteiro tentando fazer o instrumento obter alguma leitura coerente sem sucesso. No final, acabamos usando um frequencimetro para descobrir que ele fazia uns poucos acessos à EPROM e em seguida parava.

No dia seguinte, o Ruy mostrou como usar o osciloscópio para ver que depois do reset ele lia a primeira instrução (depois de ler o SP e PC iniciais, é claro) e parava ao tentar acessar a RAM no endereço do SP. Descobrimos que havia um pequeno erro de projeto na parte que responde DTACK para o processador. O Roberto rapidamente corrigiu isto, e ele passou a acessar todos os 16 megabytes. Vimos que ele estava encontrando uma instrução ilegal, e ao tentar atender esta exceção ele encontrava outra instrução ilegal (na verdade a mesma) e assim por diante. Cada vez o stack aumentava em três palavras e acabava enchendo todo o espaço de memória a cada poucos segundos. Pelo menos deu para ver que a decodificação do mapa de memória estava boa. Quando o asembler finalmente funcionou (acho que foi na quarta ou quinta feira) deixamos de

ter problemas com instruções ilegais. Quinta feira já estavamos gerando sincronismo horizontal e vertical: primeira execução correta de um programa. O Rury notou um erro na pinagem do conector de video e isto foi logo corrigido. Sexta feira tentamos fazer o Merlin se comunicar com um Ego sem sucesso.

Na terça (segunda foi feriado) seguimos outra sugestão do Rury, instalando oito LEDs que o programa poderia usar para sinalizar eventos (o que compensaria, em parte, a falta de instrumentos de teste). Depois de conseguirmos transmitir a mensagem "hello world!!" para o Ego, nos concentramos em testar a RAM, ~~Assumpção~~ /.

15/86 - Construção do Hardware II

No teste da RAM só podiamos ver muitos falhos no video. Gastei muito tempo para poder mostrar na tela do Ego os erros que estavam sendo encontrados. As falhas pareciam ter alguma relação com o endereço e colocamos capacitores de tântalo para melhorar a alimentação dos multiplexadores de endereço. Em seguida reforçamos a alimentação dos RAMs ~~propriamente~~ também sem nenhum efeito. Trocamos os capacitores das memória por tântalo sem afetar a sugeira que aparecia na tela. Na sexta-feira resolvi seguir o conselho do Romualdo e troquei as memorias da Hitachi por outras da Oki que eu tirei da placa de um Ego. O numero de erros diminuiu sensivelmente e o video ficou quase limpo. No sabado eu comprei cinco 74S153 para tentar aumentar a velocidade dos multiplexadores de endereço. Ao se colocar estes chips no Merlin, entretanto, a RAM praticamente parava de funcionar (provavelmente não estão pifados, mas não os

testei em outro circuito). Segunda feira eu reescrevi o programa de teste para aceitar comandos do Ego. Os comandos eram: testar a memória com um valor dado, reprogramar um registrador do 6845 e alterar uma posição de memória. Só na terça o programa funcionou perfeitamente. Eu pensei também pensei em reprojetar a seleção de memória para que a RAM não funcionasse a uma temperatura tão alta, mas a resfria-las com freon não notamos nenhuma mudança e concluí que tal alteração seria certamente infrutífera. Esta semana, devido ao feriado de primeiro de maio, só trabalhamos até quarta feira. Já descobrimos que uma dos falhas que ocorre no video é quando o 6845 coloca 1s nos ~~dez~~ nove endereços menos significativos (excluindo AØ). Outra coisa que deu para perceber com o protótipo é que os monitores que geralmente são usados com o Ego não conseguem mostrar uma tela totalmente branca, e que no modo 400 linhas com "interlace" as imagens piscam muito. Temos ainda muitos problemas por resolver.

22/6/89 - Resumo das Atividades

Muita coisa aconteceu (e muitos outros, infelizmente, deixaram de acontecer) desde a ultima anotação, como é fácil imaginar pela diferença de datas. O projeto mudou de nome, sendo agora Cristal, e não mais Merlin. Mas é melhor tentar seguir uma ordem cronológica.

Ao longo do mês de maio de 1986 foram resolvidos a maior parte dos problemas de hardware. O Alfredo escreveu um programa emulador de terminais para o PC que transmitia arquivos para o Merlin I, cujo software foi melhorado para aceitar "S-record"

gerados pelo assembler. Toda a parte de entrada e saída era feita por "polling" e a parte de interrupções nunca chegou a ser testada. Como a carga de programas era lenta demais (1200 bps) para programas de tamanho razoável, os LEDs foram substituídos por uma interface com a porta paralela do PC. O Alfredo acrescentou esta opção de carga de programas ao seu emulador, mas a diferença de desempenho não foi significativa.

Em junho decidimos criar uma firma de desenvolvimento, a Inova Tecnologia e Informática Ltda, que terminaria o projeto e o licenciaria à Softec para fabricação. Resolvemos reprojetar o hardware para apresentá-lo na feira de Informática Nacional em agosto. Foi decidido que esta versão teria cor e rede local Ethernet (Cheapernet, na realidade). A idéia de se usar 68020 foi abandonada por razões de custo. Foi definido que este novo protótipo não seria ainda a versão definitiva, mas deveria poder ser ainda bastante alterado. Em função disto, e do pouco tempo disponível, o projeto foi dividido em duas partes: a memória, que seria feita em placa de circuito impresso, e o resto, que seria feito em wire-wrap. Cada parte foi feita numa placa diferente (pois o software Smartwork tinha uma limitação de tamanho) e o sistema final ligava uma placa em cima da outra por conectores "euro", resultando um sistema compacto e mecânicamente robusto. O Rui teve a idéia

de na placa de baixo (de wire-wrap) fazer na realidade uma placa de circuito impresso de face simples ligando só as alimentações dos chips e os poucos componentes passivos. Todas as outras ligações eram feitas em wire-wrap do lado dos componentes usando pequenos pinos soldados próximos a cada pino dos chips. Decidimos construir simultaneamente três protótipos na esperança de que seria mais fácil descobrir problemas de montagem ao compararmos uma máquina com outra. O uso de uma PAL 16L8 deu bastante trabalho pelas viagens até a National Semiconductor para cada programação e pela falta de experiência com este tipo de componente. #Bump 3/h.

17/7/89 - Resumo das Atividades II

As placas só ficaram prontas na semana anterior à Feira, e eu e o Roberto trabalhamos (junto com o Romualdo e o Alfredo, que tentavam terminar o AT) dia e noite e o fim-de-semana inteiro na tentativa de fazer pelo menos um dos dois protótipos que foram montados funcionar. O problema que mais nos atrazou foi um mau contacto no terra de uma EPROM na máquina que parecia estar melhor. Resolvido isto, ela passou a executar programas. O "clock" de vídeo de 28 MHz derivado do oscilador principal de 14 MHz não ficou bom, e os "shift registers" do vídeo não trabalhavam direito. Substituimos esta parte do circuito por um oscilador integrado de 20 MHz. Toda a parte de vídeo foi ajustada de acordo, e o relógio do 68000 caiu de 7,5 para 5 MHz. Isto "estabilizou" a máquina, mas problemas com a memória continuaram. #Bump 3/h.

27/7/89 - Resumo das Atividades III

O software residente nas EPROMs da máquina continham um teste de memória que o Roberto desenvolveu no Merlin I. Só resolvemos os problemas de memória acrescentando resistores de terminação de linha na saída dos multiplexadores de endereços. Bsumpas/n.

18/10/89 - Resumo das Atividades IV

O software gráfico "bit block transfer" descrito no livro do Smalltalk-80 foi traduzido para a linguagem C e um simples programa de desenho que rodava no Merlin I foi adaptado para usar esta rotina. Em seguida, o programa foi "portado" para o Merlin II. Isto mostrou um erro de montagem na interface com o teclado de PC, que foi compensado alterando-se o software. O programa foi, então, expandido para aproveitar os recursos da nova máquina: Adaptamos uma simples interface com um "mouse" da Input e acrescentamos este recurso ao programa, mas o resultado não foi muito bom e desmanchamos a adaptação.

Ao ligarmos a máquina em monitores diferentes daquele com o qual vinhamos trabalhando, descobrimos que alguns deles interferiam na máquina de estados que gera o sincronismo horizontal e vertical. Isolando adequadamente este circuito o problema foi eliminado.

Foram mostrados na feira de Informática de 1986, no Rio de Janeiro, dois protótipos do Merlin II,

dos quais apenas um estava funcionando com o programa de desenhos. O outro servia para que o circuito da máquina pudesse ser mostrado ao visitante mais interessado sem que fosse necessário desligar a máquina que estava mostrando os desenhos (o que faria com que eles ~~os perdessem~~ fossem perdidos).

Apesar de incompleto, a parte da rede local ethernet nem tinha sido terminada, o protótipo causou uma impressão favorável no público mais entendido, de forma que o saldo da feira foi positivo.

De volta à Softec, o trabalho de hardware se limitou à tentativa de trazer os outros dois protótipos ao nível de funcionamento que apresentava aquele que foi demonstrado na feira. A falta de ferramentas adequados transformou esta tarefa num demorado processo de tentativa e erro. A Andreia substituiu o Roberto, que passou para a seção de homologação da Softec. Enquanto isto, acrescentei texto (proporcional, tipo Helvética) a programa de desenho e escrevi um "debugger". Este debugger foi usado para testar o controlador de rede local 7990, mas os resultados foram erráticos: quando frio ou muito quente não funcionava, mas enquanto esquentava parecia ficar bom. Outros aspectos da rede, como a fonte isolada de -9V, estavam sendo testados separadamente. Por causa destes problemas, e pelo fato que só adiantaria resolve-los se tivessemos uma placa ethernet ~~para~~ no PC, decidimos construir uma placa de rede local para o AT. Como o AT tem a linha "master", não precisariamos colocar uma memória "dual-port", o que complicaria muito a placa, como uma placa

de PC precisaria. Trabalhamos na placa, toda em wire-wrap, em novembro e dezembro de 1986, quando não havia ninguem trabalhando com ethernet no Brasil. O modo "master" se mostrou difícil de ser usado e, antes que conseguissemos alguma coisa com a placa, o projeto AT foi reativado na Softer de modo que perdemos o uso da máquina. A demora exagerada para se ter comunicação entre os protótipos e o sistema de desenvolvimento, e a descoberta de um erro de projeto (o 7990 foi projetado para ser ligado num barramento assíncrono do 68000 mas na nossa máquina o 68000 acessava a memória "sincronamente" por causa do vídeo. Este esquema furava com o 7990, que funcionava ou não dependendo de sua temperatura) fácil de ser corrigido no papel mas difícil de adaptar nas máquinas já construídas fez com que decidíssemos alterar um dos protótipos (o que funcionava melhor) para que aceitasse placas de expanção de PC. A primeira tentativa não deu certo por que nosso controlador de floppies não funcionava no modo "I/O". Com o acréscimo de uma "simulação de DMA" e alteração no debugger estávamos lendo discos de PC na máquina no fim de janeiro de 1987.

Foi iniciado o projeto do Merlin III, agora com o 68020. Enquanto isto, o Marcos Alexandre começou a trabalhar na parte de software do Merlin II. Nosso objetivo era colocar o smalltalk que eu havia escrito em C, e tinha começado a testar no Analix, no protótipo com a imagem virtual

tirada do Smalltalk-80 do Macintosh. Acrescentamos um winchester na máquina, e após o Marcos se familiarizar com o software que já havia sido feito, começamos a testar o gerenciador de memória do smalltalk. Como o programa rodava diretamente na máquina, sem sistema operacional, ficou muito difícil modificá-lo. O Marcos interrompeu este trabalho para escrever o Merlin DOS, muito simples mas que facilitaria o desenvolvimento de programas no protótipo.

Pudemos usar novamente o AT, mas quando testamos a placa de rede local o computador nem funcionava com ela instalada. Não descobrimos o defeito (nem como ele aparecer sozinho) e abandonamos definitivamente este projeto.

No projeto do Merlin III eu fiquei interessado num "chip-set" que a VLSI technology lançou para o Acorn Risc Machine mas, não querendo repetir o fiasco dos chips da Motorola, preferi ficar com o 68020 até que outros pessoas lançassem produtos com estes circuitos, garantindo que seu fornecimento seria viável. Humpf.

23/10/89 - Resumo das Atividades V

A arquitetura do Merlin III ficou definida assim: tres placas (cpu, memória e vídeo) ligados por um barramento próprio de 32 bits e a cpu seria também ligada a periféricos de PC por um barramento de 8 bits. O barramento de 32 bits foi depois trocado pelo Nubus quando o lançamento do Macintosh II nos chamou a atenção para este padrão.

Quando o 'Merlin-Dos e seus principais utilitários ficaram prontos, voltamos a debugar

o gerenciador de memória do interpretador de Smalltalk. Depois de discussões com o Rui e o Gabriel, eu decidi que seria interessante portar o Analix para a máquina, o que garantiria um certo mercado inicial para o hardware e daria maior tranqüilidade financeira para o desenvolvimento do Smalltalk, mais complexo e demorado. Assim, enquanto a Andreia construía o protótipo 68020, o Marcos escreveu um gerador de código 68000 para o compilador C do Analix e eu um assembler 68000 compatível com o formato objeto Analix. O Marcos também fez um "linker" novo para acomodar pequenos diferenças que fizemos neste formato.

Pronto o compilador, resolvemos esperar eu "debugar" o protótipo do Merlin III pois ficaria muito complicado colocar o Analix no Merlin II pela sua arquitetura incompleta e por restrições do 68000. Só a placa da CPU foi construída e, enquanto eu e o Alfredo tentávamos faze-la funcionar (a Andreia estava de férias e, em seguida, iria sair da Softec), o Hélio resolveu abandonar completamente o Analix. Estudamos alternativos (como o SOX) mas nenhuma era viável. O Hélio optou pelo desenvolvimento de um sistema próprio. Eu observei que já estávamos desenvolvendo um sistema (o Smalltalk) mas ele queria compatibilidade com Unix e, se possível, com Analix. O sistema deveria ser baseado no Analix mas não deveria aproveitar nenhuma linha sequer deste. Eu avisei ao Hélio que não teríamos condições de dar suporte para dois sistemas diferentes, e que um Unix-like acabaria nos impedindo definitivamente

de terminar o Smalltalk. Ele optou pelo Unix-like, e eu fiz uma especificação do sistema. Estudando as especificações, ele perguntou se não ficaria muito lento no AT. Eu expliquei que não rodaria no AT, mas só no Merlin III. Ele insistiu que era fundamental que o sistema pudesse ser usado no AT. Eu argumentei que neste caso, com o 386 à vista, o Merlin III perderia a razão de ser e ele concordou, de assim o projeto foi cancelado. Assumpção

28/10/89 - Resumo das Atividades VI

Em Novembro de 1987 o projeto Merlin acabou se transformando num sistema operacional para o 286. Foi definido que seria escrito num C orientado a objetos (tipo Objective C) e organizado como um conjunto de administradores (pequenas tarefas) em torno de um núcleo mínimo: uma evolução da estrutura do unalix (QNX).

Nesta época a Adele Goldberg veio ao Brasil e deu uma palestra sobre sua nova firma, a Parcplace Systems. Esta firma foi criada especialmente para divulgar a linguagem Smalltalk. Fiquei impressionado com o interesse da platéia por programação orientada a objetos e mesmo pela linguagem. Também achei interessante que a implementação da Parcplace roda em cima do Unix. No final, ~~conversei~~ discuti com a palestrante sobre a possibilidade de se portar o Smalltalk para o sistema operacional que estávamos fazendo. Estaria, assim, matando dois coelhos com uma cajadada só: faríamos o sistema que o Helio queria e teríamos as vantagens da idéia original com a linguagem Smalltalk. Assumpção

31/10/89 - Resumo das Atividades VII

No fim de 1987 a Acorn lançou o Archimedes, um computador com o "chip-set" risc que havia me interessado no início do ano. Era, agora, viável construir uma máquina que não fosse nem muito lenta (68000) e nem cara demais (como estava saindo o 68020).

Por volta de fevereiro de 1988 o Marcos terminou as ferramentas de suporte no analix (gerenciador de versões) e iniciou o sistema operacional, propriamente dito. Limitações no compilador de C do Analix fizeram com que tudo fosse mudado para o DOS, onde podia ser usado o Turbo C. Assim, acabamos tendo um sistema de desenvolvimento misto Analix/DOS. Várias ferramentas rodavam no DOS: preprocessador das construções orientadas a objetos, "linker" especial e gerador de classes. Hasmyp Jr.

25/11/89 - Resumo das Atividades VIII

Em Março de 1988 anunciei minha decisão de continuar o projeto da máquina Smalltalk nas minhas horas de folga. Usaria o processador RISC da Acorn/VLSI Technology.

Enquanto isto, o sistema operacional para o AT estava ficando bastante adiantado. Por volta de julho, entretanto, a especificação da parte do sistema que associa nomes aos objetos, e o problema de se copiar objetos de uma tarefa para outra a cada envio de mensagem estavam atrapalhando muito o desenvolvimento. Interrompemos o projeto para

rever estes aspectos. Para aproveitar o tempo parado eu pedi ao Marcos que escrevesse um simulador do 86C010 (Acorn RISC Machine). Eu escrevi um assembler para este processador.

Nesta época, ~~[illegible]~~ propus ao Jim Anderson da Digitalk que ele me licenciasse o fonte do Smalltalk/V (isto é: a parte escrita em Smalltalk e que é distribuída aos usuários) para que eu pudesse portá-lo para minha máquina. Ele achou que isto seria muito difícil de se fazer, mas nunca chegou a dar uma resposta final sobre o assunto.

Terminado o simulador do Marcos, avaliamos qual seria o desempenho de um emulador de PC para a nova máquina, e o resultado foi um pouco melhor que um PC de verdade. Levando estes resultados ao Hélio, insisti que a Inova deveria fazer a máquina Smalltalk, seu objetivo original. Ele disse que não via mercado para um produto assim. Propus comprar 13% das suas ações para poder tomar de fato as decisões da empresa. O Hélio avisou que tem por princípio só participar de empreendimentos dos quais tenha controle completo, mas ofereceu vender-me a parte dele da Inova.

Em Novembro de 1988 eu achei os anais da OOPSLA 86 e 87 na biblioteca da matemática (IME) e fiquei muito impressionado com a linguagem SELF e com "Smalltalk with Exemplars". Isto, além das discussões sobre a padronização do Smalltalk me levaram a abandonar a compatibilidade total com esta linguagem. Resolvi basear meu sistema no SELF e no interpretador de Smalltalk descrito em 26/2/85 neste caderno (páginas 13 a 21).

As negociações sobre o fim da Inova se arrastaram por eu não ter condições de assumir a compra da

parte do Hélio, que não me traria nenhum benefício prático. O Marcos trabalhava num editor de textos para não ficar sem fazer nada. Eu tinha um esboço do hardware pronto na forma de um coprocessador para o PC (outra sujestão do Rui). Propus terminar o sistema operacional para o AT e entregá-lo à Softec para compensar os gastos que esta tivera com a Inova e o Hélio pediu um tempo para se decidir (ele queria ver quais seriam os efeitos da nova Lei de Software sobre a importação de produtos como o QNX).

No fim de Dezembro de 1988 o Marcos foi para a Itautec e a Inova acabou na prática. Como em Março de 1989 o Hélio ainda não havia se decidido sobre a minha proposta, eu fiz outra: terminado o desenvolvimento do meu novo sistema eu faria uma versão para o 386 e a daria para a Softec para dar a situação da Inova por encerrada. Ele aceitou prontamente esta ideia, mas disse que era melhor esperar para que se passassem cinco anos para fechar a Inova (fica bem mais fácil).

Assumpção/r.

17/2/90 - Resumo das Atividades IX

Em maio de 1989 ficou óbvio que o projeto não iria muito longe sem nenhum recurso e tendo a maior parte do meu tempo gasto com o curso de microeletrônica na USP. Enviei uma descrição do projeto para o prof. Mamona do LSD e para o prof. Zuffo do LSI para ver se eles se interessariam pela ideia. Era mais importante que a máquina fosse terminada

do que ela fosse minha. O Zuffo ficou interessado e, surpreendentemente ofereceu de me dar plena participação no projeto. Na entrevista com o Takeo a idéia foi reforçada: o grande interesse do LSI era a sua maior divulgação, e não retorno financeiro. Combinamos que eu ficaria com direitos sobre o software, e o laboratório sobre o hardware.

Fiquei muito impressionado com os recursos do LSI, muito além dos de 1982-84 ou da Softec: vários analisadores lógicos modernos, emuladores 68020, muitos micros e muita gente. O prédio estava em expansão e estavam sendo aguardadas várias estações SUN. Assumpção

18/3/90 - Resumo das Atividades X

Comecei a trabalhar no L.S.I. no fim de maio de 1989. Inicialmente fiz um estudo comparando os processadores 86C010, 34010 e o transputer. Este último foi logo rejeitado por seu custo elevado. Entre os outros dois o ARM (86C010) saiu-se um pouco melhor que o 34010, mas acabei escolhendo este último pois seria usado em outro projeto no Laboratório e eu poderia aproveitar as ferramentas de desenvolvimento. Além disso, é bem mais simples usá-lo como coprocessador do PC.

Em julho eu tinha um esboço detalhado do projeto mas estava difícil acertar a configuração: rede local embutida ou em placa auxiliar. O Takeo me chamou para uma conversa e disse que havia gostado mais da idéia original do RISC. Eu lhe expliquei o raciocínio da escolha do 34010 e ele disse que este era um projeto completamente separado e não deveria ter a preocupação de

colaborar com outros projetos do L.S.I.. Ele também pediu que eu justificasse a implementação na forma de coprocessador. Eu cheguei à conclusão de que uma estação separada seria realmente melhor: por menos de 50% a mais no custo a pessoa teria dois computadores que podem ser usados por duas pessoas simultaneamente. Além disto, para quem não tivesse já um PC a economia seria enorme.

Já que compatibilidade com o resto do Laboratório não era problema, resolvi usar a rede Arcnet, mais barata que o Ethernet. Apesar de operar a apenas 2,5 Mbit/s, é uma rede eficiente e está sendo lançada a Arcnet Plus, totalmente compatível, mas de 20 Mbit/s. Alguns detalhes de instalação do Arcnet me incomodavam muito, no entanto, e eu descobri que placas Arcnet no Brasil eram mais caras que as Ethernet! De qualquer maneira, meu pai conseguiu uma amostra do COM90C65 (controlador Arcnet p/ PC) e comprou o 86C010 (processador RISC) por 39 dolares, o 86C110 (controlador de memória) por 20 dolares e o 86C310 (gerador de som e video) por 20 dolares. No início de agosto eu já tinha os principais componentes para construir um protótipo.

O segundo semestre da escola foi muito puxado, o que reduziu bastante o ritmo do trabalho. Eu não consegui informação suficiente sobre o híbrido do Arcnet: HY9068, mas o projeto quase ficou pronto. A grande complicação era a interface I^2C do teclado e mouse. Não ficou bom um projeto discreto e o projeto com o "gate-array" programável 2064 também estava complicado. Quando chegou a Feira de Informática em setembro de 1989 não havia nada para ser mostrado. Aproveitei para entrar em contato com fabricantes de placas Arcnet. Ao ver, no entanto, a rede Amplos operando no laboratório, a rede de estações SUN e o

trabalho da BRISA, resolvi mudar a rede do projeto para o Cheapernet novamente.

No fim de outubro vi que valia a pena usar um 87C751 como controlador I^2C pois em produção seria substituido pelo 83C751 (que custa 2 dólares ao invés de 20). Assim, em novembro, o projeto estava pronto com um controlador de rede 82590.

Um interpretador para a linguagem estava sendo escrito no PC em Turbo C. A arquitetura segmentada do 8088 complicava bastante o interpretador, de modo que quando as estações SUN se tornaram disponíveis eu transferi todo o desenvolvimento para elas, apesar de achar o ambiente de desenvolvimento do Turbo C um pouco melhor. Em 1988 eu havia tido a idéia de fazer a linguagem orientada a objetos ser visual com animação. Os constantes de uma expressão apareceriam como ícones no primeiro quadro, e as mensagens seriam mostradas como "menus" sendo acionados. A analogia com o uso da máquina é muito importante. A idéia foi descartada na época como pouco prática: uma equação como $3+(7*x)/y$ é muito mais simples de se compreender do que meia dúzia de ícones "pulando" na tela. O que me permitiu ressuscitar a idéia foi a percepção de que eu poderia criar objetos de alto nível, como "equação", e criar "código" animado de fácil compreensão. Uma expressão pode usar uma planilha eletrônica, um gráfico ou qualquer outra "mini linguagem" que melhor expresse a solução do problema.

Em dezembro o nome do projeto foi mudado para eLSI: estação do LSI. Diversos fatores (desculpas) fizeram com que o projeto esteja hoje na mesma situação que estava em dezembro: estudos para pedido de bolsa para o CnPq para o projeto de um microprocessador orientado a objetos, falta de automóvel, trabalho para o SIBGRAPI'90 e a espera por "data sheets" do chip Ethernet da NCR.

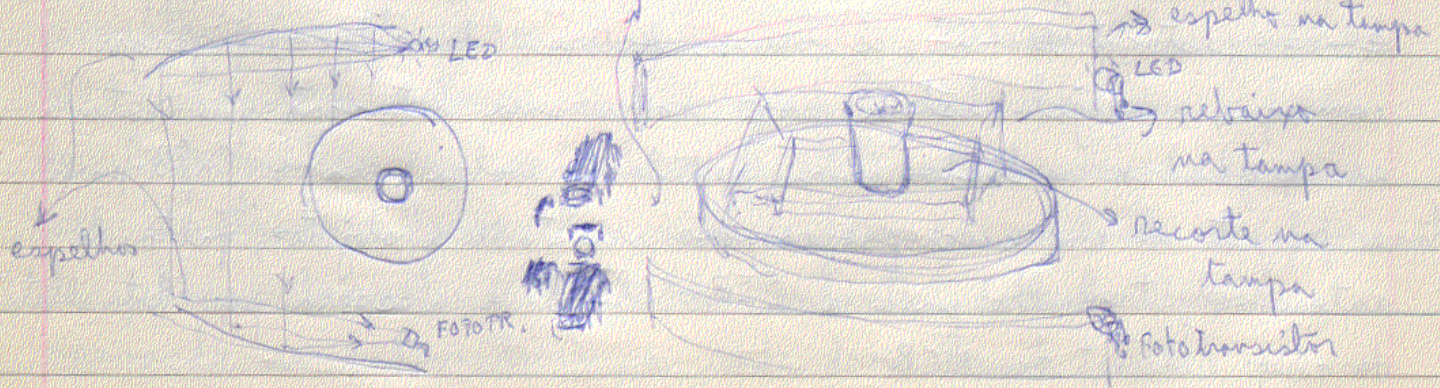
No início de fevereiro percebi que perderia tempo demais com detalhes de implementação do interpretador da linguagem. Instalei o Smalltalk V/286 no 386 que está ligado às estações SUN e vi que este era um bom ambiente de desenvolvimento que já cuidava do gerenciamento de memória e "coleta de lixo". Havia um problema de salvar a "imagem", mas encontrei uma correção para isto no BIX. Assumpção

4/3/90 - "Ponteiro" para Micros Portáteis

Em um micro portátil não é possível usar um "mouse" como "ponteiro", isto é: para controlar o cursor. Uma alternativa bem óbvia é o "trackball".

O dispositivo descrito aqui é uma pequena alavanca que pode ser movimentada com o dedão ou então segura entre o dedão e o dedo indicador. Ela está fixada num disco que desliza sobre um plano, e pode se movimentar dentro de um pequeno quadrado que representa a tela. A posição da alavanca dentro do quadrado representa diretamente a posição do cursor na tela.

O disco interrompe, em cada eixo, a luz entre um LED e fototransistor. Como mostra a figura, dois espelhos semiparabólicos em cada eixo fazem com que os feixes de luz sejam perpendiculares ao eixo de deslocamento. A luz que chega ao fototransistor é proporcional à posição da alavanca neste eixo. placa



A grande vantagem deste projeto é a robustez mecânica e facilidade de produção em massa. Como é analógico fica um pouco difícil estimar sua resolução, mas eu certamente não gostaria de ter que desenhar com esta coisa! Para muitas aplicações, no entanto, este dispositivo deve ser adequado. /Bsumpas/r

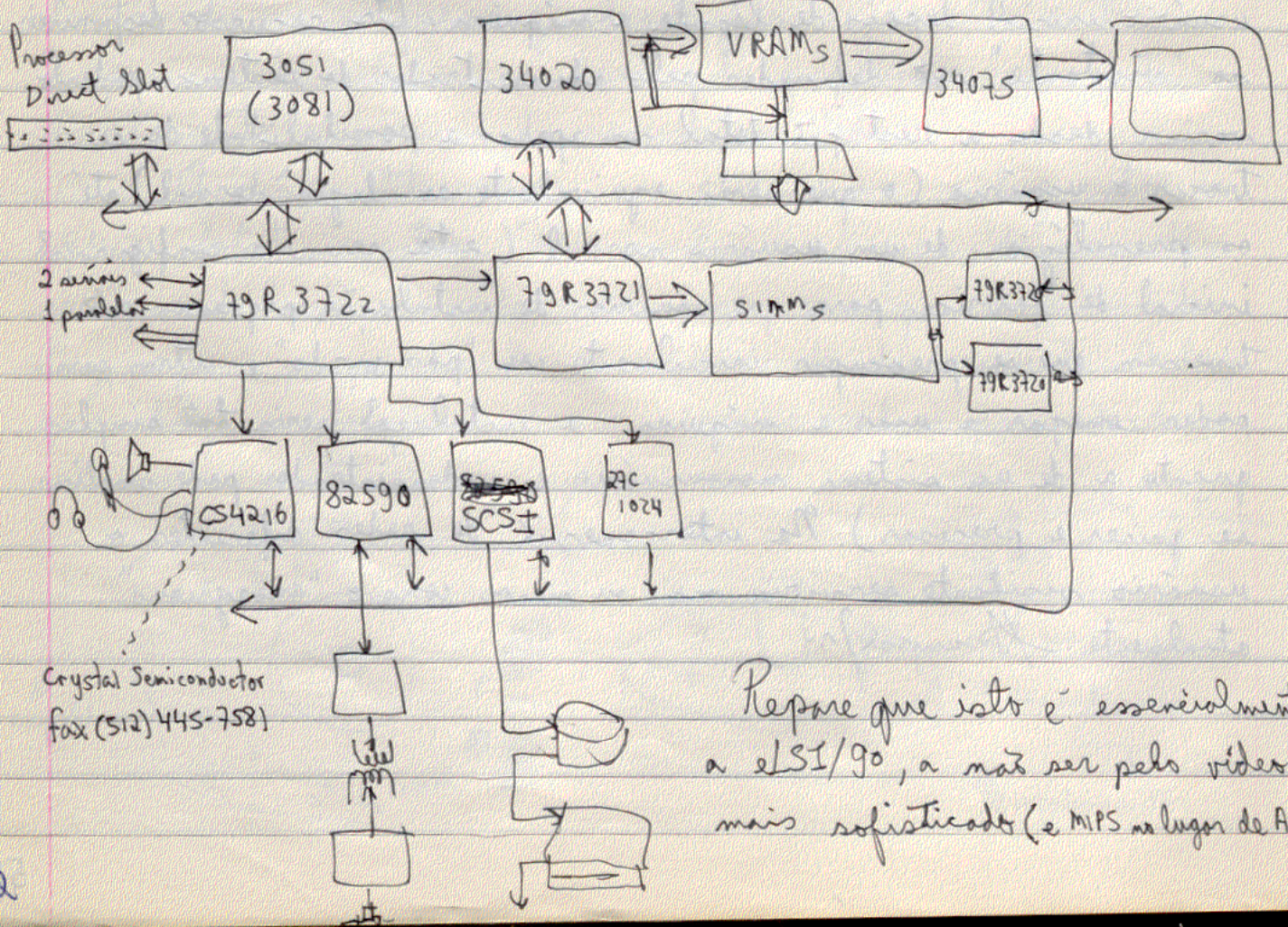
8/3/90 - Usuários Visitantes

Os sistemas multiusuários apresentam no terminal (ou na tela da estação) um pedido de identificação do usuário. Só depois de ser "aberta" uma sessão é que pode ser usado algum recurso da máquina. Muitas vezes seria interessante se pelo menos alguns recursos estivessem disponíveis mesmo antes do "login". A idéia aqui é que sempre haveria um usuário associado ao "terminal" e haveria a possibilidade de se trocar a identificação (como o comando su do Unix). Um usuário especial, o "visitante", seria o usuário inicial depois de ligada a máquina. Os recursos disponíveis ao "visitante" são definidos pelo administrador do sistema, podendo variar desde a restrição total com apenas a possibilidade de se trocar o usuário (o que seria equivalente ao "login" normal) até os previlégios de um usuário normal (esta seria a configuração inicial de fábrica para que usuários de instalações pequenas não tivessem que se preocupar inicialmente com "passwords" e outros para poder começar a usar a máquina - a instalação seria tão simples quanto a de um sistema monousuário e o administrador pode complicar se quiser e precisar). Na interconexão de redes diferentes o usuário visitante serviria mais ou menos como o "anonymous" atualmente. /Bsumpas/r

18/5/92 Não deu muito certo o formato muito rígido adotado neste livro. Havia sempre a preocupação de se "capricher", o que levava a rabiscos em outros lugares com a idéia de passar a limpo aqui (o que acabava não acontecendo). Vamos, então, mudar isto: a única regra é que a data deve aparecer com um certo destaque - fora isto vale tudo.

CPUs p/ Merlin: até 8/90 Acorn Risc Machine 86C010
8/90 - 4/91 Multi Sparc (projeto proprio)
4/91 - 11/91 Transputer T400, T800 e T9000 (cadê?)
11/91 - 5/92 i860XR (p/ gráficos 3D)

Este mês o Rui, da Deico, arranjou informações sobre o MIPS R3000 (especialmente R3051, R3081) e como eles estão interessados em padrão ACE estamos vendo se fazemos algo junto. A evolução futura p/ R4000 é interessante, mas o ACE está se defasando por que a SGI comprou a MIPS. A vantagem que tem o 860 é a facilidade de por vários processadores. Já o R3051 tem um "chipset":



Repare que isto é essencialmente o eISI/90, a não ser pelos vídeo mais sofisticados (e MIPS no lugar de ARM).

Espero que isto não saia caro, pois já tem DECstation por $ 3995!

Software: → Modelo de Objetos SELF
Memória Virtual de Objetos (LOOM)
Modelo de Paralelismo: objetos ativos, distribuídos
→ Compilador Adaptativo
GUI 3D (artificial reality): editores x agentes
Multiusuários: proteção

Merlin 1995 → desktop ⇒ portable ⇒ notebook ⇒ slate ⇒ ?

a solução é um computador realmente compacto, que possamos vestir: um óculos que projeta uma imagem enorme acoplado a uma câmera que permite ao computador "ver" tudo o que o usuário vê. Assim, o computador percebe os movimentos da cabeça do usuário e pode "sincronizar" os objetos virtuais que projeta com o ambiente onde se encontra. A câmera também pode ver as mãos do usuário, o que permite uma interface por gestos sem a necessidade de luvas esquisitas. Qualquer imagem do mundo real pode ser facilmente transferida para o virtual, sendo a câmera poderoso dispositivo de entrada. Podemos ter teclados virtuais ou escrever com canetas virtuais, mas o reconhecimento da fala seria o complemento ideal para os gestos. O importante é o computador estar à disposição o tempo todo por não ser necessário "levá-lo" a qualquer lugar (como nos portáteis) e participando das atividades do usuário.

20/5/92 Algumas aplicações diferentes para este tipo de máquina: poderíamos calcular o valor de uma compra no supermercado automaticamente ao se "scanear" os preços do produto (com optical character recognition, é claro); poderíamos captar a imagem de alguem que nos é apresentado e guardá-la

junto com outros dados numa base de dados; um estudante poderia captar a imagem da lousa durante uma aula e fazer suas próprias anotações em cima dela, podendo mais tarde "passar a limpo".

Como Vender Objetos? O próprio estilo de programação do Merlin encoraja o comércio de software "leve" e não gigantescas aplicações de $1000. Mesmo hoje, existe um bom mercado para fonts, clip-art, macros e stackware (hypercard no Mac). Várias maneiras têm sido tentadas para distribuir estas coisas: shareware, domínio público ou juntar um monte deles e vender o pacote inteiro de maneira tradicional. No Merlin há a complicação adicional da reusabilidade — se alguem usa seus objetos (mesmo que modificados) em uma nova aplicação e a vende você também devia receber algo.

Um possível esquema de proteção para objetos: cada objeto pertenceria a um usuário e poderia ser particular (não accessível a terceiros), visível (outros pessoas podem ler e copiar, mas não modificar) ou público (todos podem alterar). Isto evitaria muitos acidentes se objetos importantes pertencerem ao "superusuário" mas forem visíveis (novos objetos só podem ser criados como cópias de outros existentes, seria desagradável se estes protótipos fossem alterados ou eliminados).

Uma aplicação poderia pertencer a um usuário fictício para fins de proteção e contabilidade. Se usarmos um objeto com uma aplicação nova será feita referência a um usuário não existente em nosso sistema. Neste momento o software pode ser adquirido automaticamente (na instalação do novo "usuário") e seu valor lançado numa "conta de software" a ser paga cada mês ou a cada dois meses. A idéia é que shareware muitas vezes não é pago por ser muito inconveniente mandar $5 para algum endereço qualquer. É claro que algumas pessoas se acham espertas não

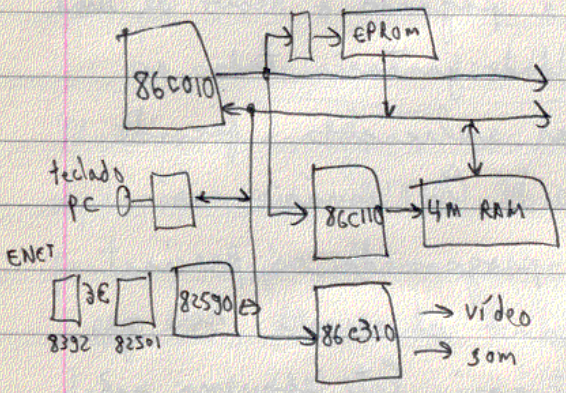
pagando já que ninguem vai ficar sabendo, mas outras pagariam se a coisa fosse facilitada. Assim, o administrador do sistema periodicamente revisaria as últimas aplicações instalados eliminando as que tivessem se mostrado inadequadas e pagando o resto de uma só vez para uma única entidade. Esta entidade juntaria todas as contas e pagaria os que desenvolveram as aplicações (ficando com uma taxa de administração, obviamente). Os usuários só precisam enviar um cheque por mês e o programador só precisa receber um cheque por mês, o que viabiliza o sharewave. Note que a questão do suporte não fica resolvida aqui. Este esquema não seria de maneira alguma obrigatório: na instalação do usuário fictício seria lido um objeto "forma de pagamento".

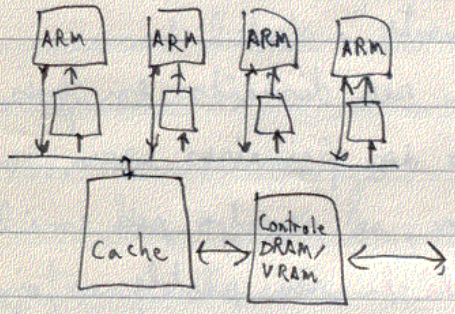
Os objetos de uma aplicação poderiam ser divididos em duas categorias: "viewer" e "editor" onde o uso dos primeiros seria gratuito mas dos outros seria pago. Assim, se recebo por email uma carta com um font diferente eu posso lê-la sem problemas, mas se quiser usar este tipo em algum lugar devo pagar. Se comprar um jogo, por exemplo, com um objeto música, devo pagar só pelo jogo. Mas se quiser editar o objeto música este deve ser pago separadamente. Assim posso revisar objetos dos outros e distribuí-los sem preocupações.

Que tal pagar pelo <u>uso</u>? A idéia de um "taxímetro" rodando não combina muito bem, na minha opinião, com computadores pessoais e programas interativos, mas sim com mainframes e processamento em "batch".

<u>Grupos</u>: uma implementação interessante para grupos seria se estes fossem mais um caso de usuários fictícios com seus próprios objetos e uma lista de usuários membros. Os usuários membros são tratados como o próprio "usuário" grupo durante acesso aos objetos do grupo (análogo ao setuser do Unix para execução de programas). Como podemos criar um grupo "todos", não precisamos dos objetos públicos mencionados acima, mas apenas dos particulares e dos visíveis.

13/7/92 O projeto do hardware voltou a ser com o ARM já que tive problemas para obter informações sobre o MIPS e resolvemos aproveitar o que temos à mão. Essencialmente temos a Elsi 90 corrigida e com teclado PC no lugar do I^2C. Pode ser encarada como merlin II com ARM. Depois, é claro, podemos usar o 86C020 para um desempenho tipo 486. Como vamos escrever todo o software podemos já incluir a possibilidade de explorar paralelismo. O novo ARM é uma célula ASIC da VTI de apenas 2,8 x 2,8 mm. Poderíamos projetar o chip ao lado para o terceiro modelo competir com o 586 em desempenho por uma fração do custo. Por outro lado, fica aberta a possibilidade de outros processados graças à portabilidade do software.





Um logbook como este ajuda muito o desenvolvimento do hardware (ajudaria mais se fosse usado direito), mas como documentar adequadamente o software? O formato dos métodos no Self é inconveniente; precisamos de algo mais genérico e flexível. Também precisamos de controle de versões.

20/7/92 Estive estudando software de controle de versões como SCCS e o que eu e o Marcos havíamos feito na Inova. Tem um artigo interessante na Byte Jan/92 e, também, "Managing the Evolution of Smalltalk-80 Systems" de Steve Putz em "Smalltalk-80: Bits of History, Words of Advice". A idéia é ter uma estrutura baseada em "bug reports", "changes" e versões. Em paralelo, um sistema de arquivamento diferencial permite o acesso a todas as versões dos fontes. Já uma lista

de "changes" constitue a diferença entre versões do sistema.

O problema é que isto é interessante para manter a documentação de desenvolvimento, que não interessa (muito) para o usuário. Veja que existem pelo menos mais dois tipos de documentação que devem ser gerenciados de maneira diferente: a documentação de componentes para permitir a reusabilidade e a documentação de módulo que permite o uso da "aplicação".

Como organizar preferências ou configurações? É claro que vamos aproveitar o esquema de herança multipla do Self. Os tipos de preferência que já identifiquei são cinco:

- de objeto: tipo de letra, cor, tamanho
- de "aplicação": volume do som, ~~função~~ operação do cursor
- de usuário: língua preferencial
- de máquina: resolução da tela, compilador
- de instalação: data e hora, usuários e permissões

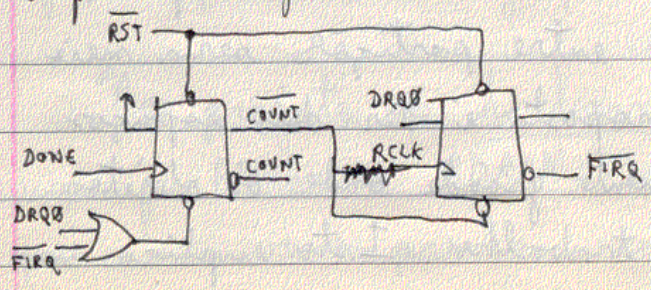
Note que as preferências são objetos complexos e não meros textos a serem interpretados. Outra coisa: a mesma preferência pode ser especificada em mais de um nível; neste caso vale a ordem indicada acima.

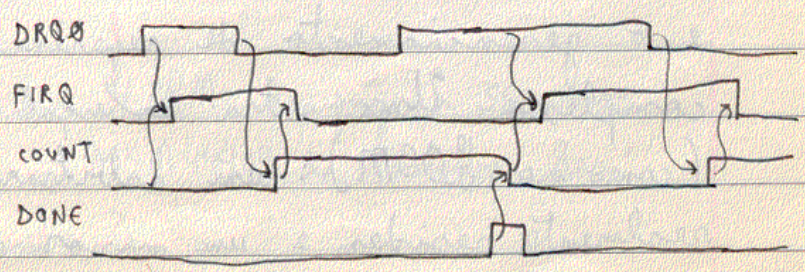
O hardware do Merlin IV não usa PAL para facilitar sua produção. As duas equações mais chatas de substituir da eLSI são:

FIRQ = DRQØ * $\overline{\text{COUNT}}$ + FIRQ * $\overline{\text{COUNT}}$ e COUNT = FIRQ * $\overline{\text{DRQØ}}$ + COUNT * $\overline{\text{DONE}}$ que

implementam, junto com o contador de 8 bits, o mecanismo de "bus throtling" para reduzir o "overhead" do nível físico da rede local.

O que temos agora é:





Uma das idéias que sempre fizeram parte do projeto do software do Merlin é a de tirar os primitivos da máquina virtual e passá-los para a imagem virtual. Assim, eles podem ser encarados com métodos pré-compilados e ter todas as características dos métodos compilados: eles podem ser criados a tempo de execução e podem chamar quaisquer outros métodos. Este mecanismo permite usar o Self do Merlin como um sistema operacional capaz de rodar programas escritos em outras linguagens. Isto implica que pode haver chaveamento de tarefas no meio de um primitivo, e eles devem ser escritos levando isto em conta. O maior problema é encaixá-los no esquema de herança, já que todos os objetos têm acesso a todos os primitivos no Self de Stanford (o mecanismo mais restritivo do Smalltalk seria mais fácil de simular com o nosso, apesar de ser ainda mais diferente).

A Memória Virtual sempre representou um ponto crítico no projeto do Merlin por ser um dos aspectos que não pode ser alterado depois (pois define o formato dos discos ou, no mínimo, a interface com os servidores de disco). Uma das razões pela qual o ARM havia sido descartado é o seu limite de 32 MBytes de memória virtual (corrigido no ARM600). Mesmo o limite de 4 GBytes não seria folgado. O esquema proposto era de dividir os objetos em espaços independentes ("partições") e usar endereçamento relativo dentro das partições para que estas pudessem ser "instaladas" em qualquer endereço virtual. A soma dos tamanhos das partições instaladas simultaneamente deve ser menor que 4 GB e o gerenciamento de referências entre partições seria meio complexo. Uma outra solução proposta é ter dois espaços (como no LOOM): um arbitrariamente grande onde os objetos realmente residem e um menor de trabalho. Isto implica em

um "overhead" de tradução nas paginações mas permite que até o ARM seja usável e elimina a pressão para migrar para 64 bits. Poderíamos usar compressão para aproveitar melhor os discos neste esquema, mas tenho dúvidas sobre como isto funcionaria. Temos que incorporar a este mecanismo segurança (tanto proteção entre usuários e grupos como técnicos para evitar acidentes que possam comprometer a integridade da imagem virtual) e um bom "garbage collector" distribuído.

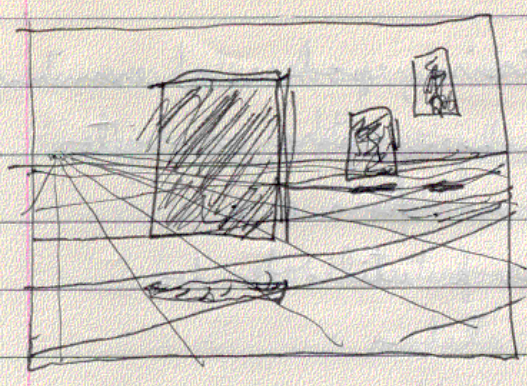
11/8/92 Alguns artigos que achei sobre a Memória Virtual, todos de Paul Wilson da Universidade de Illinois, Chicago: "Design of the Opportunistic Garbage Collector" em OOPSLA'89, "Demonic Memory for Process Histories" in 89 Conf. on Programming Language Design and Implementation e "Pointer Swizzling at Page Fault Time: Efficiently Supporting Huge Address Spaces on Standard Hardware" numa Sigarch razoavelmente recente. O "pointer swizzling" gera dois espaços, como no Loom, mas aproveita o hardware de paginação para evitar checagens constantes no software. Já a "demonic memory" permite recuperar qualquer estado anterior de execução, o que resolve o problema de segurança mencionado anteriormente. Com este tipo de recurso (se for possível implementar, o que não é muito provavel) existe um "undo" infinito a nível de sistema e é viável "debugar" um programa executando-o para trás do ponto em que ocorreu um erro. Isto talvez elimine a necessidade do "garbage collector" para objetos velhos pois eles raramente são alterados e não existe sentido em "recuperar" espaço em um Worm (talvez o "GC" ainda seja util para ~~exitos~~ agrupar objetos que costumam ser alterados).

No projeto de hardware com 4 ARMs da página 56 vamos usar o RAMBUS e RDRAMs como a MSM44409 da Toshiba. Será que ainda vamos precisar do cache?

24/8/92 Oops! A MSM44409 é na realidade uma CDRAM da Mitsubishi. A RDRAM é fabricada pela Toshiba, Nec, Fujitsu e outras, mas ainda não tenho os números dos componentes.

Uma simplificação do esquema de proteção de objetos é ter apenas um tipo: pertence a um usuário e pode ser lido e alterado por este usuário e apenas lido pelos demais usuários. A idéia é que um objeto só pode realmente ser lido de for referido por algum objeto acessível ao usuário. A "raiz" do grafo de objetos, o objeto "sessão", é sempre invisível aos outros usuários. Para que um objeto possa ser lido por outros, deve ser colocada uma referência em algum objeto público. Note que isto torna visível todos os objetos referidos por este. Dado ciclos no grafo de objetos existe o perigo desta operação tornar visíveis indevidamente muitos, ou todos, os outros objetos. Só com exemplos práticos poderemos saber se isto é um problema real ou não. Quanto a outros poderem modificar objetos de um usuário, isto pode ser resolvido com a estrutura de grupos descrita na página 55.

2/9/92 Na interface com o usuário podemos restringir o movimento a um plano apesar da interface ser 3D. Considerando o dispositivo de entrada como sendo um trackball vamos adotar a convenção de que se o usuário continuar girando a esfera quando o cursor já está na borda da tela é uma indicação para alterar o ponto de vista. Na borda de esquerda provoca uma rotação à esquerda, na direita rotação para a direita, na superior faz andar para frente e na inferior para trás. O ponto de vista está sempre a uma certa altura de um plano que se estende ao infinito. Os objetos flutuam sobre o plano



projetando uma sombra sobre o mesmo como se iluminados por cima. Não é muito comum, em nossa experiência em ambientes 3D, olhar para cima ou para baixo. Tais movimentos tendem a desorientar o usuário em interfaces como simuladores de vôo ou de modelagem.

30/11/92 O Hardware do Merlin IV foi construído e está sendo depurado. O layout foi feito no PADS pelo pessoal da DataMOS e fabricada a placa de 4 camadas na Itaucom. As RAMs e alguns outros chips só devem chegar dia 10/12. O primeiro erro descoberto foi a falta de uma trilha no "pull-up" do ABE do processador, o que fazia com que este não gerasse endereços. Os latches de endereço estavam com os controles trocados e ontém fizemos uma adaptação para corrigir isto. O ganho de 10 para o amplificador do microfone não dá nem para começo - deve ser aumentado para 250. Os resistores de bias estão bem errados: deviam ser 2.2 MΩ. A realimentação das saídas de som foi feita pelo positivo: mais uma adaptação a ser feita.

Vai ser lançado o Controlador de Vídeo novo, o 86C630, em janeiro. Este trabalha com altas resoluções e até 24 bpp e também controla LCDs. Isto é muito conveniente para o Merlin IV, que pode ser tipo laptop.

O Software Básico do Merlin vais ser uma versão reduzida do Self ao invés de um debuger/monitor tradicional. Está sendo escrito um programa que lê um arquivo no formato Self extendido para que métodos possam ser escritos em assembler e gera um arquivo binário (para ser gravado nos EPROMs) com os objetos

e rotinas e um pequeno interpretador. Assim, poderemos examinar e alterar registradores e memórias com mensagens para os objetos certos. Novos comandos podem ser criados na hora e a característica de polimorfismo pode estender a utilidade dos comandos básicos.

14/1/93 Continuamos a eliminar pequenos erros do Hardware do Merlin IV. Há um erro no netlist que impede o funcionamento de placas de expansão lentas. Isto foi deixado assim mesmo por enquanto. O inversor do sincronismo vertical foi passado para o horizontal para que o monitor recebesse pulsos negativos (o horizontal precisa ser isolado para que o cursor funcione corretamente). Os pinos $\overline{LCH}$ e $\overline{RCH}$ estão trocados no esquema do 86C310 por erro no DataBook. Além de consertar isto, vamos colocar um diodo no circuito de reset para descarregar mais rapidamente o capacitor. Muito ruído está interferindo na parte de som: o ideal seria que o terra e V_{cc} desta parte fosse separada do resto do circuito. O que podemos tentar agora é colocar alguns capacitores e ver o que acontece.

Foram escritos três programas símples de teste em assembler ARM. O HTEST era só um monte de nops e um ~~bra~~ branch para o início. Os nops exercitavam D_0 - D_8 e A_2 - A_{10}. Já o MTEST inicializava o mapa de memória, o chipset e entrava num loop testando a memória. Depois, o programa passava a esperar uma tecla entre testes da memória. O DRAW1 faz com que o cursor (na forma de uma seta) possa ser movimentado pelo teclado. Um retângulo pode ser capturado e copiado em outro lugar. Também são gerados alguns sons.

22/1/93 O programa Draw2 faz o Merlin falar "good morning Vietnam!" além do que já fazia o Draw1. Isto encerra a fase de teste do Hardware passando à programação

de verdade (apesar de faltar a parte da rede ainda). Podemos dividir o software a ser escrito em quatro camadas:

- aplicação
- interface com usuário
- modelo gráfico
- self

A camada self é a mais básica mas inclui: interpretador e compilador adaptivo, gerenciamento de memória física e virtual com sistema de versões (demonic memory), múltiplos processos com objetos ativos e distribuídos, proteção para múltiplos usuários partilhando o mesmo espaço de objeto, objetos básicos e primitivos, inicialização e configuração do sistema.

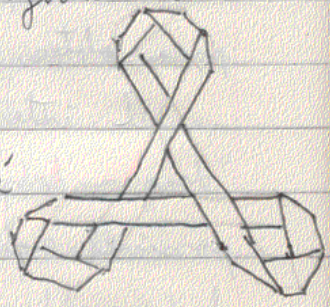
O modelo gráfico tem como maior inspiração o padrão Renderman. A idéia é fazer um sistema realmente orientado a objetos e também aproveitar elementos do Postscript, Tex, Metafont e outros. Deve haver um meio simples de definir superfícies cobertas com padrões complexos de textos e figuras dentro do próprio modelo gráfico. A implementação desta camada segue a mesma filosofia da anterior: modelo simples e ideal com cálculo adaptativo e uso extenso de "caches" para obter a melhor aproximação possível dado a capacidade da máquina.

A interface com o usuário é baseada em três operações: "drag-n-drop" que permite deslocar objetos até outro lugar com uma possível ação dependendo do local onde o objeto for deixado, "clone, drag-n-drop" que move uma cópia do objeto mas mantém a semântica da operação anterior, e o "link" que parece a operação anterior mas deixa uma referência do objeto e não uma cópia. O visual do "drop" é um pouco complicado por a interface ser 3D. A interface do pessoal de Stanford é um ponto de partida interessante mas inclui muitos botõezinhos "anônimos" (mais de quatro tipos). Seria mais interessante dar um "clone, drag & drop" do objeto "implementor" para um slot de método para que a cópia se transformasse na resposta

(um vetor de objetos que implementam o método) do que acionar um botão sem nenhuma marcação. Para uma segunda operação do mesmo tipo podemos aproveitar a resposta anterior (que também é um objeto tipo "implementors") e arrastá-lo até outro slot. Outros aspectos da interface são as lentes de aumento e o uso de multiplas vistas (editores) para cada objeto. Entre os objetos que podem ser vistos nas salas 3D são outras salas (para as quais podemos passar) formando um "hyperespaço". Uma sala especial aparece sempre na mesma posição na tela e representa o próprio usuário (ou melhor, sua sessão). Ela pode servir, entre outras coisas, como "bolso" para levar objetos de um lugar para outro no "hyperespaço".

Pelo código das camadas inferiores e o uso de objetos persistentes, as aplicações devem ser realmente muito pequenas. Na verdade objetos que podem ser considerados "aplicações" em certa fase poderão ser caracterizados cada vez mais como de "sistema" na medida que outras aplicações fizerem uso deles. No Merlin a distinção aplicação/sistema pode ser feita como objetos que não têm o seu "protocolo" padronizados e os que têm.

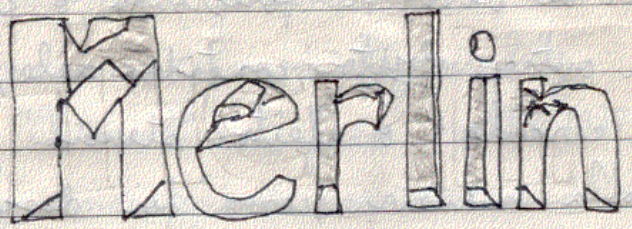
25/1/93 Uma idéia para um <u>símbolo do Merlin</u> é



que é uma variação um pouco mais sofisticada do "Möbius Strip" que é uma única superfície sem fronteiras no espaço 3D. Ela também lembra um pouco Escher por parecer estar se aproximando ao se seguir seu desenho no sentido anti-horário apesar de emendar na origem. A idéia a ser transmitida é a da concretização do impossível.

Uma maneira visualmente marcante de se escrever Merlin seria com letras azuis escuros e bordas amarelos indicando que as letras são recortadas na página; estrelas no fundo azul

fazem a palavra parecer uma abertura que dá vista para outro universo. A figura do lado ficou com a perspectiva exagerada, mas a idéia é esta.



A idéia para o Modelo Gráfico é ter dois tipos de objetos: os de modelagem (3D) e os de layout (2D). Os objetos de modelagem descrevem a forma básica de superfícies e sólidos. Uma superfície pode ser detalhada por objetos de layout que delimitam retângulos sobre a superfície. Para cada ponto da superfície o objeto de layout pode ser interrogado se o ponto pertence ao "foreground" ou ao "background". O objeto pode conter outros objetos de layout e passar a decisão para um deles, pode usar uma expressão em Self para achar a resposta ou pode passar a decisão para um objeto de layout localizado abaixo deste. Uma vez classificado o ponto, é feito o caminho inverso até achar um objeto que tenha "shaders" do tipo indicado. A cor do ponto da superfície é, então, calculado pela combinação dos "shaders" apropriados. A vantagem desta "herança" de shaders é o controle exato da aparência: todos as letras de um texto, por exemplo, podem ter sua cor alteradas simultaneamente sem afetar outros elementos que tiveram sua cor especificada.

Talvez fosse interessante ter outros tipos de shaders além de "foreground" e "background", como "outline". Isto evitaria que um círculo vermelho de borda azul tivesse que ser descrito com dois objetos sobrepostos.

No caso do círculo, citado acima, o objeto devolve "foreground" (talvez "outline"?) no caso do ponto cair dentro do círculo e passa o problema para o objetos que estiver abaixo em caso contrário. Isto dá um efeito de "transparência" no resto do retângulo. Se no

lugar de devolver "foreground", o círculo delegasse o problema para outros objetos contidos nele, veríamos estes objetos delimitados pelo círculo: o efeito "steneil" do Postscript.

Da maneira como foi descrita até aqui parece que o status de cada ponto é determinado por um método monolítico em cada objeto de layout. Na realidade, se este método for decomposto em outro de maneira padronizada se torna possível o uso de "caches" para melhorar o desempenho. Assim, um caracter em princípio tem uma expressão Self que calcula certos pontos em função de alguns parâmetros, passa curvas complexas por estes pontos (estilo Metafont) e usa estas curvas para decidir o status de um ponto. Na prática, é interessante guardar as curvas em um "cache" já que os parâmetros não mudam de um ponto para outro. Para determinada resolução, um "bitmap cache" poderia pré-calcular o status de todos os pontos.

O problema deste modelo é filtrar as coisas de maneira adequada e eficiente.

A mesma hierarquia de objetos do modelo gráfico é usada pelas mensagens da interface com o usuário de forma que a divisão mostrada na página 63 é apenas uma seqüência de desenvolvimento e não a organização real do sistema.

4/2/93 No modelo gráfico a idéia de "outline" é interessante mas não leva em conta bordas de larguras não-infinitesimais. Se no lugar de "background" e "foreground" a resposta fosse um número de 0 a 1 poderíamos fazer alguma espécie de filtragem, mas teríamos que amostrar retângulos e não pontos. A vantagem de se amostrar retângulos é que os objetos gráficos podem se adaptar à resolução da imagem que está sendo gerada. Vamos tentar fazer tudo com aritmética de ponto fixo para não encarecer o hardware.

O Software Básico do Merlin é baseado num modelo de processos descrito em "Concurrency and Reusability: from Sequential to Parallel" de Denis Caromel em Joop Sep/Oct 1990, Vol 3 No 3. A idéia básica é a do sincronismo por necessidade para maximizar o paralelismo: um processo que envia uma mensagem para outro pode continuar executando enquanto não precisar da resposta deste. Cada processo é dividido em dois objetos: o atravessador (proxy) que é quem os outros objetos do sistema enxergam como sendo o processo e a atividade que é um objeto que herda de "mixins process" e tem um método "live" e uma pilha de execução. Quando um processo envia uma mensagem para o atravessador, ele devolve um objeto "futuro" como resposta e empacota a mensagem num objeto e o envia (possivelmente através da rede) à atividade. As mensagens ficam numa fila na atividade até o seu método live mandar executar e, depois, substituir o objeto futuro pela resposta. O gerenciamento disto tudo não é fácil: se o live chega ao fim a atividade "morre" mas outros processos podem ainda querer enviar mensagens. Por outro lado, se este processo se tornar lixo ele precisa se "suicidar" de uma maneira razoavel. É fácil ver que a primeira vez que um atravessador recebe uma mensagem a sua atividade precisa ser escalonada em algum processador. Na hora de desligar este processador precisamos suspender o processo.

O núcleo do Merlin é apenas um tradutor de eventos de hardware para eventos de software. O mínimo de processamento possível deve ser feito no próprio núcleo, mas para uma mensagem de software seria necessária a criação de alguns objetos (o futuro, a própria mensagem). A solução pode ser uma lista de objetos pré-criados managed especialmente.

7/2/93 Um modelo de dispositivos de entrada muito interessante é apresentado em "Design for Supporting New Input Devices" de Michael Chen e Frank Leahy em no livro "The Art of Human - Computer Interface Design". Eles avaliam os modelos do Macintosh, GKS e NeWS e propõe uma combinação e extensão destes: eventos com um espaço de nomes extensível e estruturado em dispositivos lógicos. Quando uma aplicação não entendesse o nome de um evento, ela passaria uma lista priorizada de dispositivos lógicos e o dispositivo converte o evento para um destes e o reenvia à aplicação.

Arquitetura do S.O.

A arquitetura atual do Merlin não está boa. Ela combina elementos dos Open Systems, Microkernels e Proxy em termos de estruturação. A maioria dos objetos são **passivos**, mas alguns formam **servidores** que são extensões ativas de objetos normais. O efeito exato de uma mensagem depende do receptor, mas a semântica é sempre igual em função do sincronismo por necessidade. Um objeto servidor encapsula a sincronização pois, ao receber uma mensagem, coloca esta numa fila para ser recebida explicitamente mais tarde e devolve um objeto **futuro** (terceiro tipo até aqui) para quem enviou a mensagem. Qualquer envio de mensagem para um futuro suspende uma tarefa e a coloca numa fila do objeto futuro. Quando o servidor responder com um objeto, este substitui o futuro e todas as tarefas que estavam esperando são acordadas.

Como os servidores se parecem exatamente como objetos normais, podemos dizer que o Merlin é um Open System. Já do ponto de vista de que o sistema está implementado quase que totalmente nos servidores ele se parece com um sistema Microkernel. Uma idéia para os servidores é dividí-los em dois objetos: a **parte ativa** (que só poderia existir na memória de uma máquina em um sistema) e um **proxy** que poderia ter cópias em diversas máquinas. Só o proxy teria conhecimento da parte ativa e seria visto como o servidor pelos demais objetos. O proxy repasssa as mensagens que recebe (possivelmente através da LAN) e cuida da criação do futuro. O grosso da implementação fica dentro dos objetos proxy e futuro. Fora esta divisão em quatro tipos de objetos, esta proposta é extremamente uniforme. Alguns servidores dos sistema são: swap managers, storage managers, partition managers e drivers. O núcleo cuida de todas as interrupções e faz o chaveamento de tarefas (cuja ativação inicial, controle, etc. estariam sob os cuidados de um task manager). É claro que alguns servidores seriam executados no nível de maior privilégio mas, fora isto, as aplicações e sistema são identicos.

Uma chamada de sistema operacional, então, é quando um objeto manda uma mensagem para um servidor especial (swap manager, digamos) mas que parece identica a qualquer outra mensagem Self. Esta "sopa de objetos de sistema" não é muito prática - como vou saber quais existem? Pode ser bolada alguma organização padrão partindo-se do objeto **thisHost**. Existe um problema parecido em relação à rede e, neste caso, o problema aparece a nível de interface com o usuário.

O que há de errado com o que foi descrito até aqui? Principalmente é o fato de que os objetos não são todos realmente iguais (alguns existem para dar suporte aos outros) mas o relacionamento entre eles se limita às relações **HerdaDe** e **Contém**. Nenhuma das duas relações é a adequada para ligar um objeto ao seu storage manager, por exemplo.

As implementações tradicionais do Smalltalk e o Self de Stanford dividem o sistema em máquina virtual e imagem. O que foi descrito acima tenta reduzir mais ainda a máquina virtual e jogar tudo o que for possível dentro da imagem. A alteração mais radical é a de transformar os primitivos em objetos da imagem e, portanto, dinâmicos e manipuláveis dentro da própria linguagem. No caso do Self, um primitivo fica interessantemente parecido com métodos compilados - para dizer a verdade eles podem ser encarados como métodos pré-compilados e chegarmos a uma interface uniforme entre a máquina virtual e estes blocos de código. Fica fácil fazer com que as primitivas possam chamar qualquer método em Self. Não só podemos escrever o compilador inteiramente em Self (com um interpretador para bootstrap) mas podemos aproveitar a máquina virtual como estrutura de sistema operacional para outras linguagens - um programa em C pode ser compilado e ligado a uma biblioteca estilo Unix (que manda mensagens Self) e ser instalado como primitiva em algum objeto que pode ser invocado facilmente da interface com o usuário. O inconveniente desta idéia é que as primitivas são dependentes de máquina, o que não é bom na imagem. A solução é ter múltiplas versões de cada primitiva e usar herança para chamar a certa, o que volta a permitir imagens identicas em todas as máquinas.

Reflexões

O importante de tudo isto é que temos atividades de sistema que são quase sempre **reflexivas**: o compilador compilando a si mesmo, um objeto tela determinando sua posição na memória (e pedindo para não ser movido por coletas de lixo), um servidor determinando seu tempo de execução, e assim por diante. Por isso a estrutura inventada na Sony para o sistema Apertos é relevante. Os objetos têm uma terceira relação, o **refletor,** que o liga aos objetos que estão por trás dele, dando suporte. Os objetos **Mirror** já têm uma função parecida, mas para permitir ao objeto controlar suas características estáticas e não dinâmicas como no caso dos refletores. O refletor representa um **meta-espaço** e contém um conjunto de **meta-objetos**. A vantagem é aproveitar o mesmo meta-objeto em mais de um meta-espaço, o que representa uma economia e melhor estruturação e é fundamental em meta-objetos com estado para efeito de sincronização.

Idéia 1 - podemos criar objetos com os campos: header, mapa e refletor. Podemos criar mirrors canônicos para os objetos se dissermos que um ponteiro que aponte para a palavra seguinte ao header equivale a um mirror em cima daquele objeto. A diferença é que mensagens para o mirror são procuradas no refletor no lugar do mapa (com o mesmo código!). Criar um mirror passar a ser de custo extremamente baixo, o que torna prático o uso intenso da meta-computação. Com um meta-objeto adequado, esta implementação de mirrors pode ser totalmente compatível com a do Self original. Por que não colocar o ponteiro do refletor no mapa no lugar de em cada objeto? É que objetos devem poder **migrar** de um meta-espaço para outro e isto não deve afetar o mapa (ou deveria?).

Idéia 2 - um objeto poderia ser: header e refletor. O mapa passaria a ser um dos meta-objetos, se bem que com um status especial. Como ficam os mirrors? O número de refletores fica muito grande, igual ao número de mapas do caso anterior se os mapas apontassem para o refletor.

Interpretador e Compilador

Mesmo usando um interpretador para o bootstrap do sistema, este tem que conviver com a máquina virtual, as primitivas e, quando o compilador passa a funcionar, os métodos compilados. Queremos que as primitivas e os métodos compilados possam ser tratados de forma identica.

O maior obstáculo ao desempenho é o tempo de busca das mensagens. Cada mapa poderia ter um cache para evitar de repetir as buscas já efetuadas. Este cache da cada mapa associaria o nome da mensagem ao método correspondente (o compilador alteraria isto para apontar para o código compilado customizado para aquele mapa) e serviria, também, para encurtar buscas de filhos deste objeto. Isto contrasta (e talvez complemente) com as PICs que formam um cache no local da chamada. Com o cache de código compilado dentro da imagem e distribuído fica mais complicado o seu controle. Podemos aproveitar e tentar preservar códigos críticos entre sessões - o compilador não precisaria começar compilando a si mesmo todas as vezes.

Uma dificuldade de códigos compilados dentro da imagem é que eles podem ser deslocados pelo coletor de lixo. Eles só podem ser referenciados, entretanto, pelos caches dos mapas e pelas PICs, e não pelo código em geral. Assim, o coletor de lixo pode atualizar todas as referências.

O esquema de compilação é adaptativa - cada método é inicialmente interpretado e, se for constatado que ele é bastante usado, ele é compilado. Se seu uso for muito intenso ele pode ser compilado novamente por um compilador melhor.

O compilador gera ainda mais objetos temporários que os programas normais - talvez deva ter sua própria área de criação de objetos coletada separadamente.

Será que o compilador deve guardar todas as informações para o debugger ou pode tentar recriá-las sob demanda? Para gerar as mesmas informações o compilador deve ter as mesmas entradas exatamente, mas não posso alterar até a estrutura do objeto antes que as informações sejam exigidas?

Isto poderia acontecer em princípio, mas no instante que uma alteração destas ocorre, o sistema de dependência deve marcar o código como não válido. Posso recompilá-lo antes de realmente efetuar a alteração e guardar as informações que serão necessárias (provavelmente para a desotimização para o debugger). Se o método não estiver sendo usado (não tem nenhuma ativação) ele pode ser simplesmente descartado. Assim, o número de recompilações para gerar informações de contextos seria muito pequeno com uma perda pequena de desempenho e um ganho enorme de espaço.

Arquitetura do S.O.

O uso de refletores pode eliminar a necessidade dos proxies. Um meta-objeto do receptor controlaria a reação às mensagens. O outro problema que os proxies resolvem, referências não locais, existe de qualquer maneira em outras partes do sistema em função da memória virtual particionada.

No lugar de fazer um objeto servidor apontar para sua pilha, poderíamos integrar as duas coisas e encarar a pilha como uma extensão do próprio objeto. Fica mais fácil sincronizar a coleta de lixo do objeto e sua pilha e fizerem parte do mesmo espaço. Parece também simplificar a passivação e restauração do servidor entre sessões. A idéia de não permitir pilhas diferentes ocuparem a mesma página leva a um desperdício de memória e disco. No hardware do Merlin IV temos apenas 128 páginas físicas. A vantagem é uma proteção maior e determinação de "stack overflow" por hardware com o uso de uma página de guarda.

Quando o coletor de lixo descobre que uma tarefa não é mais referenciada, ele manda uma mensagem especial para ela se matar. Ela então termina ordenadamente a execução e pode ser coletada (quando ela tenta voltar do método inicial sua pilha é declarada lixo). O problema é que os coletores por cópia (praticamente todos os modernos) só podem descobrir que objetos são referenciados, e não quais não são mais usados. A solução é marcar todos os servidores e desmarcar os que tiverem algum apontador. É só percorrer a lista de todos os servidores uma segunda vez mandando mensagens para os que ainda estiverem marcados. Uma mensagem é melhor que uma variável (como no Eiffel) pois se um servidor solitário estiver processando alguma mensagem durante uma coleta ele podem enviar uma referência de si mesmo na resposta. Para garantir que é lixo, um servidor deve esperar a mensagem final se repetir:

```
faz: true.
[ faz ] whileTrue: [
   firstMess == "fim" ifTrue: [
                          serve: "fim".
                          messQueue isEmpty ifTrue: [
                                                [messQueue isEmpty]
                                                        whileFalse.
                                                firstMess == "fim"
                                                        ifTrue: [faz: false].
                                              ]. "else goto loop"
                                    ]
                             IfFalse: [ ... ].
                       ].
```

É claro que se o servidor souber que ele não devolve referências para si mesmo isto pode ser bastante simplificado. Por que messQueue isEmpty? Para evitar que um processamento longo acumule mensagens "fim" de várias coletas.

Mecânica

O Merlin IV é uma placa tamanho Euro 6U (formato A5, eu acho) e o primeiro protótipo está ligado a um painel de alumínio que contém diversos conectores, um capacitor enorme e o alto falante. O teclado, padrão PC XT, é ligado por um conector. Não foi feita uma interface especial para mouse ou trackball já que a idéia é embutir um trackball abaixo da barra de espaço do teclado e refazer o firmware do teclado para mandar informações do trackball pela interface normal. As outras conexões do Merlin IV são: vídeo, transformador de 9VAC, fone de ouvido e (no painel) saída para amplificador de potência, entrada de microfone, chave liga-desliga e fusível. Um conector DIN 96 permitiria uma placa de expansão para experiências.

A placa do Merlin IV deve diminuir na próxima revisão com o uso da técnica de montagem de superficie. Não tem sentido manter a máquina separada do teclado, apesar de conjuntos integrados lembrarem brinquedos. O Merlin V será capaz de usar telas de cristal líquido e ser usado em computadores tipo “notebook”.

Arquitetura do S.O.

O artigo "Sistema Orientado a Objects Merlin em Máquinas Paralelas" enviado para o SBAC-PAD 93 descreve idéias da organização do sistema operacional e do sinergismo das técnicas adotadas.

O modelo de paralelismo foi simplificado ao tornar todos os objetos servidores. O controle de cópias é feito agora em nível de sessão: objetos de um usuário que não esteja "loggado" no sistema são garantidamente apenas de leitura e podem ser duplicados à vontade pelo sistema. Como os objetos básicos pertencem a usuários especiais, desaparece a necessidade de objetos passivos para se obter desempenho aceitável. O sistema continua aberto no sentido que alguem pode entrar como um destes usuários especiais e alterar os objetos básicos. O desempenho do sistema só seria prejudicado nessa hora (se bem que isto só devira ser feito quando ninguém mais está usando o sistema de qualquer jeito).

Um aspecto interessante que não foi abordado pelo artigo é que como muitos objetos de uma sessão são apenas para leitura, o compilador adaptativo pode encarar as "variáveis de instância" como constantes e gerar código mais eficiente.

A estrutura proposta para os objetos é: um header seguido de um ponteiro ou para um contexto ou para o refletor. O mapa seria apenas mais um meta-objeto. Caso o objeto esteja executando é o seu contexto que aponta para o refletor (na realidade para o contexto do refletor). Os refletores e outros objetos especiais podem sempre ter um contexto associado, mas a maior parte dos objetos "devolve" seu contexto entre execução de mensagens. Um objeto pode ter mais de um contexto para executar métodos recursivos. O número de refletores passa a ser muito grande, mas representa objetos que realmente fazem parte de uma família (aliviando o mapa de códigos compilados e caches e outras tranqueiras). O primeiro meta-objeto de um refletor é garantidamente o mapa para poder aumentar a velocidade de busca de slots.

Hardware

A parte da rede local está dando muito trabalho. O ARM2 se mostrou muito lento para enviar um byte a cada 800 ns sob demanda. Alteramos o clock para 36 MHz, o que vai permitir modos gráficos com mais resolução e que leva o processador a 12 MHz. Apareceram problemas de escrita na DRAM em função do uso do 74LS14 no lugar de 74F14. Colocando um 74F04 o problema desapareceu. Existe ainda um problema de acessos à ROM reprogramarem o vídeo nesta velocidade.

Quanto à rede, um ARM610 resolveria facilmente o problema. Como medida temporária, faremos um modificação para que quando for executada um instrução de coprocessador o programa só continua quando a rede pedir outro byte. Podemos testar este programa:

```
ldr r8,[r9],#4     ; carrega 4 bytes para maior velocidade
copr               ; espera a rede pedir
strb r8,[r10]      ; envia como DMA
mov r8,r8,lsr #8   ; pega o segundo byte
copr
strb r8,[r10]
mov r8,r8,lsr #8   ; pega o terceiro byte
copr
strb r8,[r10]
mov r8,r8,lsr #8   ; pega o quarto byte
copr
strb r8,[r10]
ldr r8,[r9],#4     ; carrega mais 4 bytes e começa tudo de novo
```

Esta rotina executada da RAM deve ter velocidade suficiente para a rede funcionar. O hardware proposto também libera o processador no caso de uma interrupção - a rotiina supõe sempre 1500 bytes mas termina quando o chip de rede interromper. A leitura é análoga. O problema é que interrupções de teclado ou de som podem fazer com que um pacote seja perdido. A solução é simplesmente tentar de novo nestes casos. Com 12 MHz e transeferindo um byte de cada vez sob demanda do FIRQ foi preciso desligar o vídeo para obter um byte a cada 1600 ns. Como os próximos protótipos usarão o ARM610 esta solução pode ser adotada só para este primeiro caso seja necessária.

A rotina de FIRQ fica dedicada à recepção. Quando é necessário transmitir a FIRQ é inibida e a rotina descrita acima é chamada.

Foi considerada a possibilidade de se usar uma memória de buffer para eliminar estes problemas de temporização, mas isto complicaria o projeto e não afetaria o desempenho das máquinas com o processador novo que teria um overhead igual para enviar os bytes para a memória (sob demanda do FIRQ como seria no caso de se usar o 8390 como chip de rede) ou diretamente para o 82593. Assim, o circuito projetado é o mais símples e eficiente quando o processador aguenta.

Umas coisas que podem ser acrescentadas ao hardware: slots PCMCIA, interface SCSI, interface para cristal líquido. Recebi os Data Sheets dos processadores da família ARM6, mas do VIDC novo só veio uma xerox meio vaga. Seria interessante ver se não é possível projetar uma placa universal que servisse tanto para o Merlin IV atualizado como para o Merlin V conforme que componentes fossem soldados. Isto reduziria bastante os custos.

Quanto ao Merlin VI, tenho maiores informações sobre o RAMBUS e os tempos de acesso são horríveis - da ordem de 150 ns. Talvez com dois bancos "interleaved" ainda se possa salvar a idéia.

GUI

O programa em BASIC que mostrava como seria a interface com o usuário foi parcialmente traduzido para a linguagem de máquina do ARM. Mesmo executando da RAM e usando multiplicações de ponto fixo leva pouco menos de um segundo para desenhar o fundo e o chão. Comparando com o jogo "Battle of Britain" que usa perspectiva real, o programa usando perspectiva de desenho causa uma impressão bem aceitável. Acho que é porque estamos acustumados já com a TV e o cinema.

A decisão de eliminar os objetos passivos vai simplificar bastante o projeto da parte de interface com o usuário dos programas.

Falta ainda a parte de filtragem no modelo gráfico. Uma idéia é enviar um retângulo representado o píxel no lugar de um ponto para ser interpretado pelos retângulos de textura. Sempre que a textura estivesse inteiramente contida no píxel ela devolveria uma constante como resposta. Isto também faz com que não sejam calculados detalhes desnecessários (ficar calculando o contorno de uma letra quando o parágrafo inteiro é menor que o píxel).

O pessoal do LSI arranjou software para trabalhar com o PowerGlove no modo alta resolução. Seria interessante lligar uma delas ao Merlin para demonstrações do potencial da GUI. Eles também têm uns softwares de realidade virtual. Seria interessante ver se existe fontes para entender como obter o máximo de velocidade (a mais óbvia técnica, usar apenas uma pequena janela, não pode ser aproveitada já que nossa GUI sempre ocupa a tela inteira).

MerlinSoft

Na lista de discussão comp.sys.acorn.tech um sujeito perguntou como poderia arrumar um emulador do ARM. Eu mencionei o simulador que o Marcos Alexandre fez na Inova em 88 e o assembler que eu fiz. Várias pessoas se mostraram interessadas em cópias desses programas. Corrigi alguns bugs do assembler e fiz o simulador calcular o carry flag corretamente (o Marcos havia

invertido seu funcionamento para que as instruções "SBC" e "RSC" pudessem funcionar. Acontece que a definição dessas intruções no manual da VTI de 87 estavam erradas, como mostram os data sheets que acabei de receber). Traduzi as mensagens para Inglês bem como os comentários do assembler (o simulador é puro estilo MAV - sem comentários). Será interessante se possoas já puderem ir aproveitando resultados indiretos do projeto Merlin.

+ Arquitetura do S.O.

É interessante notar que um objeto nunca deveria rodar em paralelo com seu refletor. Todos os refletores poderiam ter uma prioridade maior do que os objetos normais, que só poderiam executar quando todos os refletores estivessem parados. Fica garantida, assim, que nenhuma invocação de meta-computação vai encontrar o refletor ocupados. Também evita problemas de sincronização. O problema é que refletores também são objetos e também têm refletores. Em face disso é realmente possível atribuir prioridades para cada objeto; ou seja - existe uma árvore ou um grafo de refletores?

A implementação dos primitivos pode ser feita a nível de meta-objetos (eles seriam invocados usando reflexão) ou podem

Formato dos Objetos

A implementação do Self de Stanford (S.S.) define vários tipos de slots. O formato de um slot em um mapa é:

slot name
slot type
slot contents | slot offset | data slot index
dependency link1
dependency link2

É interessante observar os detalhes de slot type. Os bits definidos são: 2 bits de tipo, 1 bit de éVM, 2 bits de visibilidade e os bits que restam (tirando TAG) indicam a prioridade de pai. Os tipos de slot são: object, map, argument e assignment. Map equivale a constante. As visibilidades são: private, public e undeclared. Os slots tipo VM não existem realmente no S.S., mas seriam apenas slots superparticulares na implementação proposta. Seu acesso se dá apenas pela máquina virtual através de primitivas, e não pelo envio de mensagens. O tipo VM assignment não existe já que os valores dos slots são alterados em outro nível.

Vamos às definições que vinham sendo usadas para diversos tipos de objetos (devem ser alteradas em função da introdução de reflexão):

```
method: ( | <code> = ( | source. fileName. lineNum. literals <- vector. codes <- byteVector |
                ). :self*. :arg1. :arg2. temp1.... | )
vector: ( | :<length>. parent* = ??. <1>. <2>. <3>..... | )
byteVector:  ( | :<bytes>. :<length>. parent* = ??. | )
mirror:  ( | :<reflectee>. parent* = ??. primitives* = ???. | )
block: ( | parent* = ??. :<scope>. value = ( | <code> = ( | ... | ). :<lexicalParent>*. :arg1.... | )
                | )
methodActivation: ( | <receiver>. <expressionStack>. <position>. <code>. <selector>.
                <methodHolder>. <sender>. | )
blockMethodActivation: ( | <receiver>. <expressionStack>. <position>. <code>.
                <lexicalParent>. <sender>. <selector> | )
process: ( | <stack>. | )
```

Os três últimos não tem sentido na atual implementação só com objetos ativos. O mirror também deve ser bem diferente e aqui tem um slot primitives* em função de não herdar do lobby como a maioria dos objetos. A combinação :<xxxx> visava indicar uma constante residente no próprio objeto. O esquema acima foi desenvolvido para uma implementação não orientada a objetos do Self(!!!) onde o primeiro slot do mapa era testado para determinar qual o tipo de objeto que estava sendo manipulado - por isso o byteVector começa com :<bytes> no lugar de :<length> para distinguí-lo de vector. O S.S. usa outro sistema orientado a objetos, o C++, na implementação, o que deixa sua marca na forma de VTABLES nos mapas. Com os refletores podemos usar a própria orientação a objetos do Self para implementá-lo(!!!).

Convenções de Tempo de Execução

Um elemento crítico do projeto do sistema é como as rotinas usam os registradores da máquina e o formato dos objetos a tempo de execução. Os objetos de suporte de execução seriam: contextos, mensagens, filas de mensagens e objetos futuros. Em princípio os três últimos poderiam ser combinados para melhorar a eficiência. O contexto guarda os valores dos registradores quando não está sendo executado, a pilha de expressões (de tamanho fixo como no Smalltalk) e informações auxiliares.

As convenções de tempo de execução devem ser otimizadas para as primitivas e os métodos compilados. O interpretador deve se adpatar como puder às convenções já que seu desempenho não terá quase nenhum impacto na implementação definitiva.

R0	-	:self*
R1	-	:arg1
R2	-	:arg2
R3	-	:arg3/temp8
R4	-	:arg4/temp7
R5	-	:arg5/temp6
R6	-	:arg6/temp5
R7	-	<extra args>/temp4
R8	-	temp3
R9	-	temp2
R10	-	temp1
R11	-	literals
R12	-	context
R13	-	context at: stacktop
R14	-	
R15	-	firstInstruction

Para enviar uma mensagem basta colocar o receptor e os parâmetros em R0 em diante e executar SWI <selector index> que a rotina de trap guarda os registradores no contexto, alloca um objeto futuro e guarda R0-R7 nele (e [r11,<selector index>]) e chama o refletor de [R0] para receber a mensagem e modifica R0 do contexto original para apontar para o objeto futuro. Se o receptor estiver pronto para receber, o seletor é procurado e os registradores recebem os valores acima. Caso contrário, o futuro entra na fila do contexto ocupado.

Cada vez que alguém tenta enviar uma mensagem para o objeto futuro, o seu contexto é colocado numa fila do próprio futuro (o que suspende o objeto).

Quando um objeto responde a uma mensagem, o seu refletor coloca a resposta no futuro que estava aguardando, acorda todas os objetos que estavam na file deste e carrega as informações da próxima mensagem da fila (se houver) nos registradores. O problema é saber quando o objeto futuro pode ser reutilizado - as referências ao futuro podem ir sendo corrigidas a medida que forem ocorrendo ou pderiam ser alteradas todas de uma vez. A primeria alternativa complica saber quando se livrar do futuro e a segunda tem um desempenho inaceitável.

Todo este "overhead" pode ser evitado com Polymorphic Inline Caches com o custo da eliminação do paralelismo. Vale a pena, entretanto, para a maior parte dos casos.

Tratamento da Recursão

Um dos problemas mais sérios do modelo de paralelismo proposto é que como cada objeto executa apenas uma mensagem de cada vez (o que simplifica incrivelmente a programação paralela ao evitar problemas de inconsistências) mensagens direta ou indiretamente recursivas provocariam sempre deadlocks. Para resolver isto, fica definido que podem ser criados contextos "extras" para que a execução possa prosseguir. Isto viola o modelo proposto mas não introduz nenhum tipo de erro que já não exista em programas recursivos seqüenciais. A questão é como detectar estes caso para criar o contexto extra. Vejamos algumas seqüências possíveis com cinco objetos A-E que enviam mensagens em ordem e E envia uma para B. (X)#> indica bloqueado na mensagem indicada esperando que X responda, %> indica que o outro objeto não está pronto para receber a mensagem e => indica que a execução continua. Depois do ";" estão as filas com os futuros indicados por números.

A=>B; A:? B:1
A=>B=>C; A:? B:1 C:2
A=>B=>C=>D; A:? B:1 C:2 D:3
A=>B=>C=>D=>E; A:? B:1 C:2 D:3 E:4
A=>B=>C=>D=>E%>B; A:? B:1,5 C:2 D:3 E:4
A=>B=>C(D)#>D=>E%>B; A:? B:1,5 C:2 D:3 E:4 3:C
A=>B(D)#>C(D)#>D=>E(B)#%>B; A:? B:1,5 C:2 D:3 E:4 2:B 3:C 5:E
A(B)#>B(B)#>C(B)#>D(B)#>E(B)#%>B; A:? B:1,5 C:2 D:3 E:4 1:A 2:B 3:C 4:D 5:E
A(B)#>B(B)#>C(B)#>D(B)#>E(B)#>B'; A:? B:1 C:2 D:3 E:4 B':5 1:A 2:B 3:C 4:D 5:E
A(?)#>B(?)#>C(?)#>D(?)#>E; A:? B:1 C:2 D:3 E:4 1:A 2:B 3:C 4:D

Cada contexto lembra de que contexto depende para continuar (ou se refere a si mesmo se não estiver bloqueado). Quando bloqueia num futuro, altera este valor de acordo com o contexto em cuja fila está o futuro e propaga este valor para todos os contextos bloqueados em futuros que estão em sua fila. Assim, na linha 8 acima, o contexto D bloqueia em B e propaga isso para C que propaga para B etc... Quando recebe resposta, o contexto volta a apontar para si mesmo mas NÃO precisa propagar esta informação - isto deixa seus dependentes com informações obsoletas mas que serão corrigidas se ele voltar a bloquear. As informações velhas não devem atrapalhar o funcionamento do sistema (a não ser, talvez, atrasar a coleta de lixo). Sempre que marcarmos um objeto como dependente de si mesmo é criado um contexto extra e o futuro que provocou isto é colocado em sua fila. O caso mais complicado ocorre quando o contexto bloqueia e decobre que este futuro está na fila de um dependente seu - ele precisar descobrir qual futuro de sua fila que provocou a recursão.

Um problema deste algorítmo é que ele é em si recursivo - cada contexto alterado pode ter vários futuros na fila, cada um potencialmente com vários contextos. Toda esta árvore precisa ser marcada, mas a esperança é que o caso norma seja de listas símples com dois ou três elementos.

Dealocação de Futuros

Uma não solução parao problema da dealocação dos futuros é voltar a dividí-los em partes: a parte apontada diretamente pelos vários objetos apontaria inicialmente para o resto do futuro (mensagem e filas) e eventualmente para a resposta. O grosso do futuro pode ser reaproveitado imediatamente após a resposta e só o pequeno "toco" teria de permanecer alocado até o coletor de lixo se livrar dele.

Tratamento da Recursão

O algorítmo de detecção de recursão é recursivo, mas podemos evitar uso de pilhas se cada contexto que estiver numa fila de um futuro lembrar qual é este futuro. Assim é possível descer e subir a árvore de contextos bloquedos para marcar a dependência apenas seguindo os ponteiros normais. Cada contexto tem, então, um ponteiro para o próximo contexto (na fila de bloqueados, na fila de livres ou na fila de prontos), uma indicação do contexto do qual este depende e uma indicação de que fila que está este contexto.

Dealocação de Futuros

Uma variação da idéia de referência indireta de futuros é ter uma tabela com ponteiros para o futuro ou sua resolução. A página onde está esta tabela pode ser marcada como não presente para causar um trap a cada tentativa de acesso (bloqueando o processo no caso de futuro ou a eliminação da indireção no caso de objetos normais).

Blocos e Paralelismo

O modelo de paralelismo adotado sofre com os blocos. Quando estes são usados como parâmetros de uma mensagem ela deveria sempre ser síncrona para não alterar a semântica do programa seqüencial no caso de acesso global ou ao contexto léxico. O seguinte caso torna isto evidente:

```
| c |
c: 0.
x do: [ :i | c: c + 1 ].
c _Print
```

Um pouco mais sutil:

```
1 to: 10 do: [ :i | i _Print ].
1 to: 10 do: [ :z | z _Print ].
```

O problema é que _Print causa uma referência global indiretamente. Pode ser argumentado que o '.' deveria ser sempre um ponto de sincronização (como foi adotado no Smalltalk que defini em 1984). Acesso globais podem estragar a execução de mensagens com aparência bastante inocentes mas os blocos provocam o grosso dos problemas.

Um contra-exemplo:

```
'(' _Print.
car _Print.
'.' _Print.
cdr _Print.
')' _Print.
```

O que isto gera na saída só é garantido no caso de operação seqüencial. A segunda linha pode demorar mais que a terceira para executar e a referência global ao stream de saída ocorre tarde demais para manter a seqüência correta (várias mensagens de um objeto para outro são executadas na ordem em que foram enviadas). Note que o ".", que corresponde a um "pop", não aparece nos bytecodes do Self.

Já o próximo exemplo funciona sem problemas:

```
Transcript show: '('.
Transcript show: car printString.
Transcript show: '.'.
Transcript show: cdr printString.
Transcript show: ')'.
```

Assim, a solução parece ser cuidar apenas do caso de blocos na implementação e resolver os outros problemas mudando de estilo no próprio fonte em Self. Isto está mais de acordo com o que o compilador vai poder fazer quando estiver funcionando (a otimização de blocos é uma fator importante no desempenho do sistema).

Tiny Self

A idéia inicial era ter um pequeno interpretador de Self em Smalltalk rodando no PC suficiente para poder ler a descrição do sistema em Self (incluindo os trechos em assembler com alguma sintaxe especial) e gerar o conteúdo a ser gravado nas EPROMs. Isto ainda está incompleto. Comecei, agora, a escrever algo diretamente em assembler (usando o CPP para permitir #include e #define para melhorar um pouco o nível das coisas). Os objetos do TS (Tiny Self) serão definidos como uma seqüência de diretivas "dw". Tudo será gerado como se fosse rodar da RAM - um pequeno bootstrap inicializa a RAM e copia as EPROMs para lá.

Justificando o Self

Um ambiente mono-linguagem sempre foi parte implícita do projeto Merlin, primeiro com o Smalltalk e agora com o Self. O suporte para outras linguagens será possível num esquema semelhante ao usado pelo Transputer para programas não em Occam (estes programas são tratados como primitivas do Self). Nenhum outro ambiente moderno adota uma linguagem como "nativa" como era o Basic no início dos computadores pessoais. Vejamos algumas linguagens em uso:

- programação (Basic, Pascal, C, Clipper, ...)
- automação e linha de comando (Batch, Shell, Rexx)
- automação em aplicações (macros Excel, Word, 123, ...)
- composição de textos (TEX)
- composição de gráficos (Postscript)
- texturas de objetos 3D (Shaders do Renderman)

No simulador que foi escrito em Self a linguagem foi usado como linha de comando, para descrever o comportamento dos componentes e para descrever a estrutura dos componentes. Muitas descrições estáticas podem ser substituídas por programas em alguma linguagem com vantagens. Poucas linguagens são suficientemente flexíveis para desempenharem satisfatoriamente tantos papeis diferentes. Alguma variante do Lisp pode ser para muitos uma boa candidata, mas o estilo orientado a objetos do Merlin torna o Self a melhor opção (a vantagem dos Lisps é a estrutura símples e manipulável dos programas).

O uso de uma linguagem única vai permitir uma evolução mais rápida (já que todos os esforços se concentram em uma única implementação), facilidade de aprendizado e interação entre os diversos usos.

Projeto Merlin

Segue uma decomposição resumida do Projeto Merlin. Representa, de certa forma, um planejamento geral do projeto. Deve ser útil como um "checklist" para avaliar o desenvolvimento. Isto tem sido alterado ao longo do projeto e com certeza não representa a palavra final sobre o assunto.

1. Hardware
    1.1. CPUs
        1.1.1. Merlin I a Merlin III - projetos na Inova com o 680X0 de 1985 a 1987
        1.1.2. Merlin IV - protótipo com VL86C010 (ARM2) e chips de apoio com fonte de 5W com transformador de parede, entrada de teclado XT, saída de vídeo VGA, Ethernet 10Base2, saída de som estéreo para fone de ouvido e amplificador externo, alto falante interno mono, entrada de som mono, conector de expansão. Foram fabricadas três placas de quatro camadas.
        1.1.3. Merlin V
        1.1.4. Merlin IV Jr
    1.2. Periféricos
2. Software
    2.1. Software Básico

2.1.1. Testes do Hardware
2.1.2. "MetaCore"
2.1.3. Refletores
2.1.4. Meta Objetos
2.1.5. Interpretador de Self
2.1.6. Primitivas
2.1.7. Adaptação dos objetos básicos de Stanford
2.1.8. Memória Virtual
2.1.9. Compilador Adaptativo
2.1.10. Modelo Gráfico
2.1.11. GUI
2.1.12. Objetos Texto
2.1.13. Objetos Multimídia
2.1.14. Objetos Matemáticos

2.2. Sistema de Desenvolvimento
2.2.1. Assembler ARM
2.2.2. Simulador ARM
2.2.3. Smalltalk V & Smalltalk V/286
2.2.4. Sistema de versões em QNX
2.2.5. Outros da Inova - Assembler Z80, 8048, 68000, compilador 68000, Editor de Texto, software de instalação para MS DOS, Merlin DOS, Smalltalk em C

2.3. Aplicações
2.3.1. Documentos Compostos
2.3.2. Planilhas & "Workflows"

3. Divulgação
3.1. Simpósios
3.1.1. SIBGRAPI 90 - sessão poster sobre a eLSI (atual Merlin IV).
3.1.2. SBAC-PAD 92 - palestra "Supercomputador Orientado a Objetos". Descreve o MS8702, a linguagem Self e algumas idéias para a implementação do sistema nesta máquina.
3.1.3. SBAC-PAD 93 - palestra "O Sistema Orientado a Objetos Merlin em Máquinas Paralelas". Menciona de leve o hardware dando destaque à implementação do sistema no MS8702.

3.2. Eventos
3.2.1. SUCESU 86 - foi apresentado um protótipo do Merlin II com apenas um símples software de pintura

3.3. Imprensa
3.3.1 Byte Brasil 09/93 - não menciona o Merlin mas serve para divulgar a programação orientada a objetos e preparar terreno.

3.4. Vídeos

4. Infraestrutura
4.1. Administração
4.1.1 Preliminares
4.1.1.1. Encerramento da Inova
4.1.1.2. Abertura da Merlin Computadores
4.1.1.3. Registro de marca e logotipo no INPE
4.1.1.4. Curso de Administração de Pequenas Empresas

4.2. Laboratório
- 4.2.1. Biblioteca
  - 4.2.1.1. Revistas
  - 4.2.1.2. Livros de Smalltalk
  - 4.2.1.3. Livros de Redes
  - 4.2.1.4. Livros de Computação Gráfica
  - 4.2.1.5. Data Sheets
  - 4.2.1.6. Outros Manuais
  - 4.2.1.7. Outros Livros
  - 4.2.1.8. Documentação Interna
- 4.2.2. PC 386DX para desenvolvimento de software e hardware.
- 4.2.3. Programador de EPROMs STD
- 4.2.4. Apagador de EPROM
- 4.2.5. Rede Ethernet com software TCP/IP de domínio público KA9Q
- 4.2.6. Multímetro
- 4.2.7. Osciloscópio (emprestado da Arotec)

4.3. Associações
- 4.3.1. LSI-USP - A associação com o LSI no projeto Merlin começou em 1989 com a proposta do desenvolvimento conjunto do projeto. Foi feito o projeto detalhado da estação do LSI - eLSI, hoje Merlin IV. Outros projetos impediram a continuação deste trabalho, mas ficou acertado o porte do Sistema Merlin para o MS8702. O osciloscópio digital/analisador lógico HP do laboratório de hardware é usado esporadicamente para a solução de problemas complexos no protótipo. A ligação do LSI com o Internet também está sendo aproveitada no projeto.
- 4.3.2. PUC-SP - Foram escolhidos três alunos para receberem bolsas do CNPq para trabalhar em cima de multimídia para o Merlin.
- 4.3.3. Objectview - Foi estudado o desenvolvimento de um curso de Smalltalk.

4.4. Fábrica

4.5. Fornecedores

4.6. Centro de Distribuição de Aplicações

4.7. Distribuição de Hardware

4.8. Supporte

Modelo de Paralelismo

O contra-exemplo em 930803-1 está ERRADO! A mensagem "self car" da segunda linha pode demorar arbitrariamente para executar, mas o futuro que devolve bloqueia imediatamente o objeto em função do _Print. A terceira linha só vai executar DEPOIS que o car devolver um resultado que substitua seu futuro. Conclusão: os _Print são enviados na ordem em que aparecem no fonte - sempre! É claro que o resultado ainda é não-deterministico, mas o exemplo: 'x' _Print. 'y' _Print também é. Evite "double-dispatching" para objetos globais.

Object Store

O paper "Texas: An Efficient, Portable Persistent Store" do pessoal da Universidade do Texas descreve algumas idéias interessantes como o LSS - Log Structured Storage. A idéia é dividir cada disco em partições separadas para cada usuário (apenas lógicamente). A raiz de cada partição é um objeto "sala". As salas interconectadas podem abranger vários discos diferentes. Levar um objeto para outra sala significa possivelmente movê-lo para outro disco.

Os discos fixos da rede inteira são explorados implicitamente seguindo as conexões construídas. Já os floppies locais devem "aparecer" ao serem inseridos. Os floppies remotos também devem poder ser acessados, mas como é mais comum terem sido colocados por outro usuário eles devem ser explícitamente pedidos. Em princípio poderíamos deduzir que floppies remotos com partições accessíveis pelo usuário interessariam para ele, mas o mais normal é o floppy ser do usuário "visitante" ou "todos". Neste caso os usuários de uma rede grande seriam inundados com um número muito grande de disquettes.

GUI

Além do esquema de salas descrito acima, deve existir uma "sala" eu que equivale (um pouco) ao clipboard dos sistemas tradicionais. Deve ser trivial ir para esta sala e voltar para o ponto de origem carregando alguns objetos. Uma idéia é substituir a sala atual por "eu" enquanto uma tecla especial estiver apertada. Isto serve de bolso para levar objetos para salas mais distantes sem obrigar que o carregar objetos seja uma operação mais complexa (que tenha que ocorrer concorrentemente com a navegação). A sala "eu" também pode servir como raiz de objetos como os floppies mencionados acima que não estão conectados ao sistema de salas normais.

Estrutura Básica dos Objetos

Cada objeto tem um header e um ponteiro para seu contexto atual ou para seu refletor. Uma idéia é combinar os refletores e mapas em um só objeto. Também os futuros e contextos poderiam ser combinados. O map map ficaria igual ao m^{zero} e teríamos o contexto do objeto normal apontando para o contexto do refletor (mapa) apontando para o contexto de m^{zero}.

O mapa de um contexto poderia ser o mapa o código correspondente (forçando um pouco a barra). Antes de começar a execução, um contexto poderia apontar para um mapa genérico que depois seria substituído. O contexto do refletor (mapa) dos contextos seria o nosso MetaCore.

Um problema com o esquema descrito é que os objetos são alterados durante a execução (no seu ponteiro para o contexto atual) mas boa parte dos objetos é apenas de leitura. Seria bom fatorar estas alterações para que pudessemos marcar as páginas correspondentes como protegidas contra a escrita. Como as alterações só são feitas pelo próprio sistema, talvez tenhamos um meio termo onde o sistema tenha permissão de escrita mas o usuário não.

Outra dificuldade é que o formato dos contextos varia de processador para processador. A parte que varia deveria ser fatorada em um objeto separado que não seria guardada em disco. A proposta de combinar o protótipo do contexto (slots de parâmetro) com o resto das informações do código (para evitar indireções desnecessárias) vai contra esta idéia.

Execução

Uma mensagem é enviada colocando-se os valores corretos nos registradores e executando SWI. Temos as mensagens normais, mensagens para self, mensagens redirecionadas e redirecionadas para pai específico. Poderíamos acrescentar mensagens para o refletor e usar a sintaxe de redireção em Self: "mirror.migrate: persistent". Isto, é claro, dentro do próprio objeto - outros teriam que criar um mirror e trabalhar com este.

O retorno não local vai substituir o futuro (= contexto atual nesta proposta) por um objeto normal. Ser for algo como "^ 0 " isto é óbvio, mas se for "^ obj slowMess" poderíamos fazer uma otimização parecida com a de "tail recursion" em Logo ou LISP. Poderíamos reaproveitar o próprio contexto atual para enviar slowMess. A resposta, então, irá diretamente para quem estiver bloqueado em função deste contexto. Isto economiza memória, tempo e libera o objeto que estava executando para receber imediatamente outra mensagem. Uma caso meio chato é quando temos "^ instVar" e instVar já tem um futuro guardado. Esta situação implica em um "future merging" - como seria feito isto?

Estrutura Básica dos Objetos

Uma dificuldade do esquema definido até agora é a necessidade de ficar alterando o ponteiro do contexto atual de objetos que deveriam ser apenas de leitura. Seria bom se estes objetos ficassem um páginas protegidas contra a escrita. Observamos que as modificações ocorrem a nível de sistema, o que permite manter a proteção pelo menos a nível de usuário.

Um modo totalmente diferente de encarar isso é observar que um objeto apenas de leitura pode ter cópias em vários nós diferentes e, como conseqüência, vários contextos atuais ao mesmo tempo. Isto não causa conflitos em função da garantia de que o objeto não será (diretamente) alterado. Se isto é verdade, o que impede a existência de diversos contextos atuais dentre de um único nó? Na realidade, a idéia de "contexto atual" só serve como um mecanismo de sincronisação para manter a coerência de objetos alteráveis. Podemos, então, evitar a filar de mensagens a ser recebidas no caso de objetos apenas de leitura e colocar toda nova mensagem para estes diretamente no "ready queue". Logo, o objeto não precisa mais apontar para o primeiro contexto da fila (que muda a toda hora) e passamos a ter um paralelismo muito maior, além de ser uma simplificação. Resumindo: os objetos apenas de leitura apontam sempre para seu mapa/refletor e pode ter muitos contextos ativos (ou bloqueados) no sistema.

Todos os contextos vivos podem ser encontrados partindo-se da lista de contextos prontos. Na ausência de "deadlocks", todo contexto bloqueado depende (pelo menos indiretamente) de algum contextos pronto. E os contextos que estão à espera de serem recebidos dependem de um contexto pronto (o primeiro da fila, que é apontado pelo objeto).

Tratamento de Erros

No Self de Stanford, cada processo está associado a uma pilha diferente. Todas as atividades referentes a este processo estão guardadas na pilha, de modo que não é muito dificil suspender ou matar um processo. Quando ocorre algum erro, podemos passar esta pilha para o "debugger" analisar. No nosso caso, cada objeto é um (ou mais) processos. Em uma expressão como "(a/b) + c factorial" os dois termos da soma são calculados em paralelo. Se b for zero, a divisão vai provocar um erro, mas o fatorial continuará sendo calculado de qualquer forma. Isto é muito dificil de impedir, restando a alternativa de deixar correr e cancelar o resultado final. A proposta é o uso de "respostas envenenadas" cujo efeito se espalha lentamente pelo sistema. Quando uma expressão como "a/b" provoca um erro, o futuro associado é marcado como sendo envenenado (onde o veneno contém um string que descreve o tipo de erro) e todos os contextos que dependem deste futuro também são marcados. A infraestrutura para detectar recursão serve perfeitamente para esta tarefa. À medida que novos contextos tentam mandar mensagens para futuros envenenados, eles "morrem" também e espalham o veneno para os que estiverem suspensos esperando seu resultado. Isto, é claro, deixa computações orfãos (como o c factorial acima) que continuam mesmo quando seus resultados serão jogados fora quando terminarem.

Como descrito até aqui, o veneno se espalharia até contaminar o sistema inteiro. Para contê-lo, um programa deve usar um antídoto do tipo "try: expression Else: [| :type. :name. :mirror | .....]". Se expression for envenenada, o bloco é executado e type e name informam o que ocorreu e mirror é um espelho que permite manipular o futuro envenenado (e seus associados) sem nenhum risco. O resultado da expressão é o resultado do bloco. No primeiro exemplo, a solução poderia ser bastante símples (graças à semântica de blocos no Self): "(try: a/b Else: 0) + c factorial". A interface com o usuário sempre tem um comando deste tipo para isolar os "programas" dos erros dos outros. Note que try:Else: pode ser implementado em Self sem apoio especial.

Normalmente o bloco de exceção testaria type e tomaria uma ação diferente para cada caso. Em certos casos ele poderia querer propagar o veneno (ou iniciar a propagação de outro). Poderíamos ter um comando equivalente ao throw do LISP ou raise da Ada: "poison: 'user error'".

É importante manter a compatibilidade entre o mecanismo descrito e o tratamento de erros nas primitivas do Self de Stanford. Lá, cada primitiva tem duas versões: uma simples e outra que aceita um bloco que é executado em caso de erro na primitiva. No manual tem o exemplo:

"3 _IntAdd: 'a' IfFail: [| :error. :name | (name, ' failed with ', error, '.') printLine. 0]"

Error é um string descrevendo o erro e name é o nome da primitiva que falhou. Na descrição do try:Else: aparece name também, apesar disto poder ser obtido indiretamente de mirror. O try:Else: é mais econômico do que gerar duas versões de cada primitivo - talvez só os casos em que o IfFail: é realmente usado poderiam ser programados em Self, mesmo.

Manuais

Este mecanismo de documentação cronológica (continuação do caderno laranja "Merlin Inspect") tem como vantagem o fato das alterações sucessivas não eliminarem as idéias anteriores. Uma preocupação com formatação também reduziria a documentação efetivamente gerada. A necessidade crescente de descrever o projeto para outras pessoas, entretanto, exige a criação de um manual mais organizado. A versão inicial será em inglês para ampliar suas possibilidades de divulgação. Deverá ter apêndices com informações sobre o hardware do Merlin IV e do MS8702.

Convenções de Tempo de Execução

Em 930713-1, foi definida uma convenção de uso dos registradores para o ARM. Em princípio, esta convenção permitiria passar os parâmentros por registradores sem o uso da memória. Isto não ocorreria na prática, porém, em função da possibilidade de ser outra mensagem, e não a que está sendo enviada, a próxima a ter "sua vez" no processador. A alta taxa de chaveamento de tarefas faz com que os registradores acabem sendo "salvos" na memória de qualquer forma. O fato de que um método tinha de estragar seus registradores de entrada para enviar uma mensagem acaba gerando muitas instruções de LDR a mais. A solução que tinha sido adotada era a de não salvar os registradores alterados em seu contexto (eles já estavam sendo guardados na mensagem que estava sendo enviada) - depois do envio de uma mensagem os registradores voltam (quase todos) ao conteúdo do início do método. Isto, é claro, supõe um sistema não preemptivo. Se uma interrupção exigisse que outro contexto fosse ativado na volta (page fault, por exemplo) os registradores não teriam onde ser guardados. O sistema não preemptivo funcionaria razoavelmente para os programas gerados pelo compilador Self, mas como o Merlin admite primitivas arbitrárias escritas pelo usuário, pelo menos as primitivas teriam que poder ser interrompidas.

Uma solução para tudo isto é a convenção mostrada abaixo e inspirada no funcionamento do Sparc:

R0	-	:self*
R1	-	:arg1
R2	-	:arg2
R3	-	:arg3
R4	-	outReceiver
R5	-	outArg1/temp6
R6	-	outArg2/temp5
R7	-	outArg3/temp4
R8	-	temp3
R9	-	temp2
R10	-	temp1
R11	-	literals
R12	-	context
R13	-	context at: stackTop (:arg4 ....)
R14	-	
R15	-	firstInstruction

O conteúdo de R4-R7 vão parar em R0-R3 no contexto receptor. A resposta da mensagem volta em R4. Em relação anterior, esta penaliza as mensagem com 4 a 6 parâmetros. Uma estatística feita com boa parte do código do Self 2.0, porém, mostrou o seguinte resultado:

Numero de Objetos	3498
Numero de Objetos c/Slots	1295
Numero de Objetos de Dados	127
Numero de Objetos de Heranca	1252
Numero de Slots Normais	802
Numero de Slots Constantes	3939
Numero de Slots de Argum.	605
Numero de Mensagens c/ 0 Arg	14551
Numero de Mensagens c/ 1 Arg	6076

Numero de Mensagens c/ 2 Arg	916
Numero de Mensagens c/ 3 Arg	197
Numero de Mensagens c/ 4 Arg	102
Numero de Mensagens c/ 5 Arg	44
Numero de Mensagens c/ 6 Arg	8
Numero de Mensagens c/ 7 Arg	8
Numero de Mensagens c/ 8 Arg	7
Numero de Mensagens c/ 9 Arg	15
Numero de Mensagens Seq.	8724
Numero de Mensagens Par.	5730

É fácil ver que 102+44+8 mensagens afetadas não terão grande impacto no desempenho. Em processadores com mais registradores, podemos aumentar o número de parâmetros em registradores para até 7 (note que a linha acima que diz 9 Arg indica mensagens com 9 ou mais parâmetros). No caso do 68020, que só tem 8 registradores de dados (fica dificil aproveitar os de endereço), só é possível acomodar mensagens com até um parâmetro. Os dados acima mostram que mesmo isso é razoável.

Modelo de Paralelísmo

Em 930803-1 foi apresentado um problema com mensagens paralelas envolvendo blocos e mensagens indiretas para um mesmo objeto. 931022-1 indica que o exemplo dado funcionaria corretamente e que o problema é mais restrito do que parecia inicialmente. Nos dados mostrados acima, aparece uma linha com "Mensagens Seq." que indica que o resultado desta mensagem será o receptor de outra, provocando o bloqueio imediato. "Mensagens Par." indica aquelas cujo resultado será um parâmetro de outra, e que pode ser executada em paralelo. A soma dos dois tipos não bate com a soma de mensagens por número de parâmetros - existem as mensagens cujo resultado será simplesmente descartado (é seguida por um "."). Tinha sido definido que estas mensagens seriam paralelas, a não ser que um de seus parâmetros seja um bloco.

Na primeira implementação, seria melhor fazer as mensagens "de mais alto nível" seqüênciais para evitar de ficar testando se os parâmetros são blocos ou não. O impacto disto no paralelismo disponível deve ser medido mais tarde. A idéia de seqüências de códigos como pontos de sincronização já estava presente no Smalltalk descrito em 1985, e tem a vantagem de embutir menos surpresas do que o modelo mais liberal proposto anteriormente.

Tiny Self

Foi escrito o programa MKIMAGE em C que lê arquivos escritos em Self e gera vários formatos de saída e estatísticas. O formato default é um texto qie permite verificar a análise sintática. Está sendo escrita a rotina que gera saída no formato do assembler do ARM. Podem ser incluídos trechos escritos em assembler mesmo se um método estiver com a anotação "Primitive:" e seu código for só uma seqüência de textos. Estes textos podem ser códigos para vários processadores diferentes: uma linha "cpu ARM" faz com que as linhas até o próximo "cpu" sejam literalmente enviadas para a saída. As demais cpus são ignoradas neste formato (isto permite manter um único conjunto de forntes para todos os processadores). O arquivo de saída deve ser "incluído" num programa ARM com rotinas de inicialização e o resultado irá gerar as EPROMs a serem usadas no protótipo Merlin IV. A máquina dará "boot" direto no interpretador de Self. Quando a parte da rede TCP/IP estiver funcionando, a idéia é alterar o Self através de _RunScript de arquivos mantidos no

servidor de arquivos e accessados por FTP

Estrutura dos Métodos

A filosofia da implementação inicial é juntar objetos num só quando estes são sempre alocados juntos. Foi assim que os refletores e os mapas foram integrados. Também os contextos (ativações), os futuros e os objetos representando mensagens foram transformados em uma coisa só.

Uma coisa parecida acontece nos métodos: eles representam um protótipo da ativação e apontam para um objetos invisível <code> que tem informações (sempre constantes) sobre o código fonte, os bytecodes e os literais. Estes são vistos do primeiro objeto através de reflexão - provavelmente podemos dispensar o segundo objeto sem problemas.

Observando os contextos, parece que eles podem ter vários papeis: futuro, mensagem não enviada, mensagem enviada, ativação. O papel de futuro se sobrepões aos outros três, que poderiam ser distingüidos mudando-se o mapa do contexto (um caso especial de migração). A mensagem não enviada aparece no caso em que a mensagem não é gerada implicitamente pelo bytecode send, mas sim explicitamente fazendo-se um clone de "mensagem" e enviando "seletor:", "receptor:" etc. A mensagem enviada é aquela que está na fila de recepção de algum objeto e ainda tem muitos campos a preencher (literais, primeiraInstrução, etc). Quando a mensagem é executada, o seletor é procurado no receptor e seus pais e o contexto poderia adotar o refletor/mapa do código encontrado e copiar para si os valores dos slots que não os parâmetros. O mapa das mensagens enviadas e não enviadas pode ser o mesmo sem problema.

Se a ativação tem o mesmo mapa que o código, os dois seriam indistinguíveis, em princípio. O objeto código teria que ter o mesmo tamanho do contexto, o que traz dois inconvenientes. A maior parte dos slots do código não teriam valores interessantes, ocupando espaço inutilmente. Outro problema é que o formato do contexto varia de processador para processador, mas os códigos são persistentes e iguais em todas as máquinas. A solução é alterar o comportamento (inclusive o de acesso aos slots) em função dos estados: prototype, ready, running, waitForInterrupt, waitForFuture e poisoned. Observe que este tipo de objeto nunca pode receber mensagens diretamente; apenas através de um mirror, que por sua vez faz uso do "meta core" que pode ser diferente para cada tipo de processador.

Gerenciamento da Memória

Um artigo de Paul Wilson, "Real-Time Non-Copying Garbage Collection", descreve uma variação lógica do algorítmo de Baker: os objetos são marcados como estando num espaço ou em outro, mas não são movidos fisicamente (o que economiza espaço ao custo da perda da compactação automática). O custo em termos de espaço acaba sendo dois ponteiros por objeto para manter listas duplamente ligadas. Isto é muito para o Self, que tem objetos com 6 slots em média mais duas palavras de overhead (header e ponteiro para o mapa), mas evita a necessidade de se alocar semi-espaços inteiramente vazios.

A idéia é que se tenha duas listas (por tamanho de objeto - veja abaixo): uma dos objetos vivos e outra dos espaços a serem alocados. É criada uma nova lista e os objetos raizes são movidos da primeira para esta. Em seguida, esta lista é percorrida incrementalmente e os objetos apontados pelo objeto sendo examinado são trazidos da primeira lista para a atual. Quando chegamos ao fim da nova lista, todos os que sobraram na primeira lista são lixo e esta é simplesmente concatenada na lista livre.

Para conter um pouco a fragmentação, o espaço é dividido em páginas. Só objetos de um certo tamanho (potência de 2, de 16 a 512 bytes) podem ser alocados em uma determinada página. Objetos que ocupem mais de uma página são alocados em um número inteiro de páginas seguidas. Note que existem apenas 5 filas de objetos vazios: 16, 32, 64, 128 e 256 bytes. A páginas inteiras são controladas por outro mecanismo.

No Mérlin existem os objetos em ROM, que não precisam de algorítmo de alocação. Por outro lado, existem objetos que precisam ser alocados em endereços fisicos específicos para que o hardware opere corretamente. Estes são sempre vetores de bytes (puros - sem headers ou outras bobagens). Os bytes dos vetores de bytes são alocados exatamente nos mesmos esquemas dos objetos normais, mas em páginas separadas. Eles têm sua próprias filas de espaços livres.

O esquema descrito aqui poderia facilmente permitir a finalização do objetos (onde eles recebem uma mensagem antes de "morrer"), mas isto eliminária o grande benefício de liberar o espaço deles com uma simples concatenação de listas (sem percorrê-las).

Um problema meio chato do modelo de paralelismo adotado é que quando um futuro se transforma na resposta (pelo bytecode Non Local Return), podem haver referências para ele além dos contextos bloqueados (que podem ser alterados "na hora"). Como conseqüência, o futuro não pode ser liberado. O coletor de lixo, quando está percorrendo a lista do espaço novo e examinando um objeto, se observar que um slot deste se refere a um futuro de valor já conhecido (por um flag no header do futuro) elimina a indireção e resgata, se for necessário, a resposta e não o futuro. Assim, o futuro será liberado com certeza no fim desta coleta.

Note que as páginas de 512 bytes não batem com o hardware MEMC1, que usa páginas de 32K em máquinas com 4M. Isto faz com que as páginas sejam lidas em grupos de 64 do disco. Para a escrita, um esquema de marcação de cartões permite determinar as páginas de 512 bytes alteradas para que apenas estas (e as que seguem que forem do mesmo objeto) precisem ser gravadas no disco. A barreira de escrita é assim no ARM:

```
str Rx,[Robjeto,Rdeslocamento]   ; altera o slot Rdeslocamento para o valor Rx
strb Rcards,[Rcards,Robjeto>>9] ; zera o cartão correspondente
```

Observe que a barreira é de apenas uma instrução, mas custa um registrador com uma constante. O valor de Rcards deve ser um múltiplo de 256 se quisermos que a barreira armazene 0 no cartão correspondente.