SimuS + Sapiens

Manual de Uso e Descrição da Arquitetura

26 de agosto de 2017

Gabriel P. Silva
Departamento de Ciência da Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
gabriel@dcc.ufrj.br

José Antônio dos S. Borges
Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
antonio2@nce.ufrj.br

I. A Arquitetura do Processador Sapiens

A. *Apresentação*

A máquina Neander, como definida originalmente, é uma arquitetura baseada em acumulador muito simples, de caráter didático, que pode ser facilmente apresentada em poucas horas. Algumas características do processador original da máquina Neander incluem:

- Largura de dados e endereços de 8 bits;
- Dados representados em complemento a dois;
- 1 acumulador de 8 bits (AC);
- 1 apontador de instruções de 8 bits (PC);
- 1 registrador de código de condição com 2 bits: negativo (N) e zero (Z).

O Neander só possui um modo de endereçamento: o modo direto (absoluto), no qual a palavra que segue o código da instrução contém, nas instruções de manipulação de dados, o endereço de memória do operando. Nas instruções de desvio, esse endereço corresponde à posição de memória onde está a próxima instrução a ser executada.

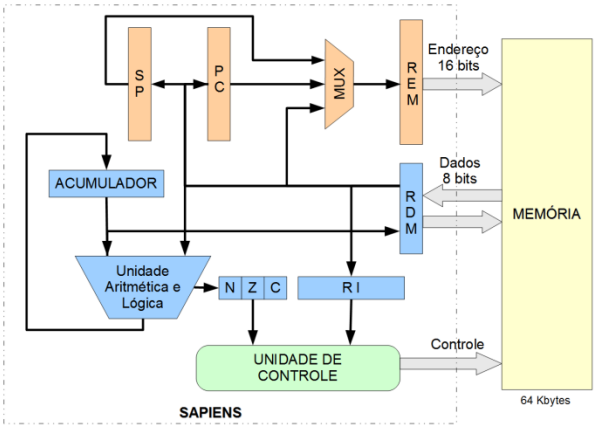

Fig. 1. Diagrama em Blocos do Processador Sapiens

O conjunto de instruções da máquina original do Neander foi estendido para incluir alguns detalhes na sua arquitetura. Essa nova arquitetura estendida, denominada Sapiens, cujo diagrama em blocos é mostrado na Figura 1, incluiu, entre outros detalhes:

- O **modo imediato** de acesso aos operandos, simplificando as operações de atribuição de dados;
- Um **modo indireto** de endereçamento, possibilitando exercitar as noções de indexação e ponteiros – que são fundamentais para entendimento de qualquer estrutura básica de programação;
- Operações de entrada e saída de dados para dispositivos E/S, em espaço de endereçamento separado da memória;
- Aumento da largura do apontador de instruções (PC) para 16 bits, permitindo endereçar até 64 Kbytes.
- Um apontador de pilha (SP, do inglês stack pointer), também de 16 bits, para possibilitar a chamada e o retorno de rotinas e procedimentos;
- Um código de condição (flag) C (do inglês carry) para o vai-um e também vem-um;
- Uma instrução de TRAP para chamada do simulador para realizar operações mais elaboradas de E/S.
- Um conjunto novo de instruções de movimentação de pilha, deslocamento do registrador, soma e subtração com vai-um/vem, entre outras.

B. *Formato das Instruções e Modos de Endereçamento*

As instruções em linguagem de máquina do processador Sapiens podem ter um, dois ou três bytes (8 bits), conforme pode ser visto na Figura 2.

Fig. 2. Formato da Instrução do Processador Sapiens

Nas instruções, o primeiro byte sempre contém o código de operação nos 6 bits mais significativos e, quando for o caso, o modo de endereçamento nos 2 bits menos significativos. As instruções com apenas um byte não tem outro operando além do acumulador, que é um operando implícito para quase todas as instruções. As instruções com dois bytes são aquelas que, além do acumulador, usam um dado imediato de 8 bits como operando no segundo byte da instrução.

Nas instruções com 3 bytes os dois últimos bytes podem conter o endereço de memória do operando (modo direto), ou o endereço de memória do ponteiro para o operando (modo indireto) ou ainda o próprio operando (modo imediato de 16 bits). Note que nos dois primeiros modos de endereçamento o operando em si possui apenas um byte de comprimento. A codificação para o modo de endereçamento é a seguinte:

00 - Direto: o segundo e terceiro bytes da instrução contêm o endereço do operando;

01 - Indireto: o segundo e terceiro bytes da instrução contêm o endereço da posição de memória com o endereço do operando (ou seja, é o endereço do ponteiro para o operando). Na linguagem de montagem, usou-se como convenção para indicar que um operando é indireto precedê-lo pela letra "@" (arroba);

10 - Imediato 8 bits: o segundo byte da instrução é o próprio operando. Na linguagem de montagem, usou-se como convenção para indicar que um operando é indireto precedê-lo pela letra "#" (tralha).

11 - Imediato 16 bits: os dois bytes seguintes à instrução são utilizados como operando. Na linguagem de montagem, usou-se como convenção para indicar que um operando é indireto precedê-lo pela letra "#" (tralha). O montador fica encarregado de gerar a codificação do operando no tamanho correto de acordo com a instrução. A única instrução que utiliza este modo é a LDS (Load Stack Pointer).

C. Códigos de Condição

A seguir são apresentados os códigos de condição do processador Sapiens, ou seja, *flags* que indicam o resultado da última operação realizada pela UAL.

- **N – (negativo):** 1 – o resultado é negativo; 0 – o resultado não é negativo
- **Z – (zero):** 1 – indica que o resultado é igual a zero; 0 – indica que o resultado é diferente de zero
- **C – (vai-um):** 1 – indica que a última operação resultou em ”vai-um” (carry), no caso de soma, ou ”vem-um” (borrow) em caso de subtração; 0 – o resultado não deu nem ”vai-um” ou ”vem-um”.

D. Descrição das Instruções

O conjunto original de instruções foi expandido para permitir uma maior capacidade de processamento. Todas as instruções originais do Neander-X foram mantidas, com exceção da instrução LDI (*load immediate*), que ainda é aceita pelo novo montador, mas na geração do programa em linguagem de máquina foi substituída pela instrução LDA #imed, que realiza a mesma função. Ou seja, um código escrito em linguagem de montagem para os processadores Neander ou Neander-X é normalmente compatível e executado sem problemas no SimuS.

Como destaque para as novas instruções introduzidas na arquitetura temos a adição e subtração com carry, (ADC e SBC), novas instruções lógicas como “ou” exclusivo, deslocamento para a direita e esquerda do acumulador (XOR, SHL, SHR e SRA), novas instruções de desvio condicional (JP, JC, JNC), instruções para chamada e retorno de procedimento (JSR e RET), instruções para a manipulação da pilha (PUSH e POP), além da própria inclusão do apontador de pilha (SP) e das instruções de carga e armazenamento do apontador de pilha em memória (LDS e STS).

As instruções lógicas e aritméticas (ADD, ADC, SUB, SBC, NOT, AND, OR, XOR, SHL, SHR, SRA) e as instruções de transferência LDA, LDS e POP afetam apenas os códigos de condição **N e Z** de acordo com o resultado produzido. As instruções lógicas e aritméticas (ADD, ADC, SUB, SBC, SHL, SHR, SRA) afetam também o código de condição **C** de acordo com o resultado produzido. As demais instruções (STA, STS, JMP, JN, JP, JZ, JNZ, JC, JNC, JSR, RET, PUSH, IN, OUT, NOP e HLT) **não** alteram os códigos de condição.

A seguir apresentamos uma tabela completa com as instruções do Sapiens e os respectivos códigos de operação.

TABELA I. TABELA DE INSTRUÇÕES

Mnemônico	Código	Descrição
NOP	0000 0000	Não faz nada
STA ender STA @ender	0001 000x	Armazena o acumulador (um byte) na memória
STS ender STS @ender	0001 010x	Armazena o apontador de pilha (dois bytes) na memória
LDA #imed LDA ender LDA @ender	0010 00xx	Carrega o operando (um byte) no acumulador
LDS #imed16 LDS ender LDS @ender	0010 01xx	Carrega o operando (dois bytes) no apontador de pilha (SP)
ADD #imed ADD ender ADD @ender	0011 00xx	Soma o acumulador com o operando (um byte)
ADC #imed ADC ender ADC @ender	0011 01xx	Soma o acumulador com o carry (flag C) e com o operando (um byte)
SUB #imed SUB ender SUB @ender	0011 10xx	Subtrai o acumulador do operando (um byte)
SBC #imed SBC ender SBC @ender	0011 11xx	Subtrai o acumulador do carry (flag C) e do operando (um byte)
OR #imed OR ender OR @ender	0100 00xx	Realiza um "ou" bit a bit entre o acumulador e o operando (um byte)
XOR #imed XOR ender XOR @ender	0100 01xx	Realiza um "ou exclusivo" bit a bit entre o acumulador e o operando (um byte)
AND #imed AND ender AND @ender	0101 00xx	Realiza um "e" bit a bit entre o acumulador e o operando (um byte)
NOT	0110 0000	Complementa ('0' ☾ '1' e '1' ☾ '0') os bits do acumulador.

Mnemônico	Código	Descrição
SHL	0111 0000	Deslocamento do acumulador de um bit para a esquerda, através do *carry*
SHR	0111 0100	Deslocamento do acumulador de um bit para a direita através do *carry*
SRA	0111 1000	Deslocamento do acumulador de um bit para a direita através do *carry*
JMP ender JMP @ender	1000 000x	Desvia a execução do programa para o endereço
JN ender JN @ender	1001 000x	Desvia a execução do programa para o endereço, apenas se N = 1
JP ender JP @ender	1001 010x	Desvia a execução do programa para o endereço, apenas se N = 0 e Z = 0
JZ ender JZ @ender	1010 000x	Desvia a execução do programa para o endereço, apenas se Z = 1
JNZ ender JNZ @ender	1010 010x	Desvia a execução do programa para o endereço, apenas se Z = 0
JC ender JC @ender	1011 000x	Desvia a execução do programa para o endereço, apenas se C =1
JNC ender JNC @ender	1011 010x	Desvia a execução do programa para o endereço, apenas se C = 0
IN ender8	1100 0000	Carrega no acumulador o valor lido no endereço de E/S
OUT ender8	1100 0100	Descarrega o conteúdo do acumulador no endereço de E/S
JSR ender JSR @ender	1101 000x	Desvia para procedimento
RET	1101 1000	Retorno de procedimento
PUSH	1110 0000	Coloca o conteúdo do acumulador no topo da pilha
POP	1110 0100	Retira o valor que está no topo da pilha e coloca no acumulador
TRAP ender TRAP @ender	1111 0000	Instrução para emulação de rotinas de E/S pelo simulador
HLT	1111 1111	Para a máquina

Foi definida uma linguagem de montagem para este processador obedecendo às regras usualmente encontradas nos sistemas comerciais e compatível com a sintaxe previamente utilizada no simulador Neanderwin. A sintaxe completa dos comandos do montador pode ser vista a seguir:

Comentários	Os comentários são começados por **ponto e vírgula (;)** e podem também ocorrer no final das linhas de instruções.
Rótulos	Um rótulo é um nome dado à próxima posição de memória. Deve ser seguido por **dois pontos (:)**.
ORG ender	A diretiva **ORG** (origin) indica ao montador que a próxima instrução ou dados devem ser colocados na posição de memória indicada por *ender*.
var **EQU** imed	A diretiva **EQU** (equate) associa um nome (rótulo) a um certo valor. Esse comando é freqüentemente usado para especificar variáveis que são posicionadas em um endereço específico de memória. Por exemplo, para posicionar a variável X no endereço 100 use: X EQU 100.
END ender	A diretiva **END** indica que o programa fonte acabou. O operando *ender* é usado para pré-carregar o PC com o endereço inicial de execução do programa.
DS imed	A diretiva **DS** (define storage) reserva um número de palavras na memória definido pelo valor *imed*.
DB imed1, imed2, imed3 ...	A diretiva **DB** (define byte) carrega nesta palavra de memória e nas seguintes o(s) valor(es) de 8bits definido(s) pelo(s) operando(s) *imed1, imed2, imed3* ...
DB imed1, imed2, imed3 ...	A diretiva **DB** (define word) carrega nesta palavra de memória e nas seguintes o(s) valor(es) de 16 bits definido(s) pelo(s) operando(s) *imed1, imed2, imed3* ...
STR "cadeia de caracteres"	A diretiva **STR** (define string) carrega nesta palavra de memória e nas seguintes o(s) valor(es) o código ASCII correspondente aos caracteres da cadeia entre aspas.

Na maioria são mnemônicos e comandos com sintaxe simplificada e de fácil utilização. A seguir um exemplo de programa em linguagem de montagem para o processador Sapiens:

```
ORG 0
INICIO:
; Faz a leitura do painel de chaves
        IN  0
        STA A
; Se a entrada for 0 termina o programa
        AND 0
        JZ  FIM
; Testa se é par (bit 0=0)
        LDA A
        AND #1
        JNZ SENAO
; Se for, faz pares++
        LDA PARES
        ADD #1
        STA PARES
        JMP INICIO
SENAO:
; Incrementa I
        LDA IMPARES
        ADD #1
        STA IMPARES
        JMP INICIO
FIM:     HLT

ORG 100
A:      DS  1
PARES:  DS  1
IMPARES:DS  1
END 0
```

E. Simulando operações de E/S usando a instrução TRAP

O SimuS tem diversos dispositivos de entrada e saída disponíveis para iteração com o usuário. Os dispostivos mais simples podem ser acessados através de operações simples de entrada e saída, com descrito a seguir:

- IN 0 : retorna no acumulador o valor binário do painel de 8 chaves;
- IN 1: retorna no acumulador o valor 0x1 quando houver um novo dado disponível no painel de chaves;
- IN 2: retorna do acumulador o valor ASCII correspondente a última tecla pressionada no teclado de 12 teclas;
- IN 3: retorna no acumulador o valor 0x1 quando uma nova tecla for pressionada no teclado de 12 teclas;
- OUT 0: o valor do acumulador é exibido no visor simples no formato hexadecimal;

- OUT 2: o valor do acumulador, que deve estar na codificação ASCII, é exibido no banner de 16 caracteres.
- OUT 3: o banner de 16 caracteres é limpo.

No sentido de ampliar e facilitar a capacidade de realizar operações de E/S introduzimos um artifício que é a utilização da instrução TRAP para isso. Em um processador real a instrução de TRAP desvia a execução para uma rotina do sistema operacional para realizar um determinado serviço ou operação de E/S para o programa do usuário. Em nosso simulador foi criado um mecanismo para facilitar a leitura e escrita em dispositivos de E/S mais complexos sem a necessidade de expor esta complexidade para os usuários do simulador.

Neste mecanismo a instrução TRAP é interpretada pelo simulador como um pedido para que ele realize uma determinada operação de E/S. O número do *trap,* ou seja, a função que está sendo solicitada é passada no acumulador. Como o processador Sapiens é carente de registradores, a instrução TRAP tem um operando adicional que é o endereço de memória onde os parâmetros adicionais necessários são passados para o módulo do simulador responsável pela realização da operação. Temos previsão para suportar inicialmente as seguintes operações:

#1 – leitura de um caractere da console. O código ASCII correspondente é colocado no acumulador.

#2 – escrita de um caractere que está no endereço definido pelo operando da instrução TRAP na console.

#3 – leitura de uma linha inteira da console para o endereço definido pelo operando da instrução TRAP.

4 – escrita de uma linha inteira na console. O operando da instrução TRAP contém o endereço para a cadeia de caracteres que deve ser terminada pelo caractere NULL (0x00). O tamanho máximo da cadeia é de 128 caracteres.

5 – chama uma rotina de temporização. O operando da instrução TRAP contém o endereço da variável com o tempo a ser esperado em milissegundos.

6 – chama uma rotina para tocar um tom. A frequência e a duração do tom estão em duas variáveis inteiras no endereço definido pelo operando da instrução TRAP.

#7 – chama uma rotina para retornar um número pseudo-aleatório entre 0 e 99 no acumulador. O operando da instrução TRAP tem o endereço com a variável com a semente inicial.

II. A Interface de Usuário do SimuS

Mantivemos a interface básica do simulador do Neanderwin, acrescentando mais algumas funcionalidades e agora destacando os módulos de E/S, que estão em janelas que são ativadas apenas quando for conveniente para o usuário. Desde o início do projeto do simulador SimuS nosso objetivo era facilitar ao máximo as atividades didáticas do professor e o apoio mais completo possível para as dificuldades comuns do aluno. Para isso foi criado um ambiente integrado para desenvolvimento, com versões para os sistemas operacionais Windows e Linux, e que inclui os seguintes módulos:

- Editor de textos integrado, que possibilita a abertura, edição e salvamento de arquivo com o código em linguagem de montagem.
- Montador (assembler) também integrado, gerando o código objeto final em linguagem de montagem. Possui compatibilidade com programas escritos para o Neander ou Neander-X.
- Simulador da arquitetura, com visualização e modificação dos elementos arquiteturais do processador, tais como registrador de instrução, apontador de instrução, apontador de pilha, acumulador, flags N, Z e C; além da execução passo-a-passo ou direta.
- Módulo de depuração, definindo pontos de parada e variáveis em memória para serem monitoradas, em caso de mudança de valor a execução é interrompida.
- Visualização e modificação da memória simulada, no formato hexadecimal os endereços e também para seu conteúdo, agora expandida para 64 Kbytes.
- Ferramenta de apoio ao aprendizado de instruções, com ajuda integrada ao editor de textos, para a geração de código em linguagem de montagem do processador Sapiens.
- Utilitário para conversão de bases binária, decimal e hexadecimal.
- Simulador de dispositivos de E/S, que inclui os tradicionais painel com 8 chaves e visor hexadecimal, acrescidos de um teclado de 12 teclas e um "banner" de 16 linhas. Estes dispositivos são acessados no espaço de endereçamento de E/S convencional com instruções de IN e OUT.
- Dispositivos especiais como uma console virtual, acessada pelo mecanismo descrito anteriormente. Com esse mecanismo outros módulos mais complexos possam ser facilmente instalados, sem necessidade de expor esta complexidade para os usuários.
- Gerador/carregador de imagem da memória simulada, podendo ser salva em formato hexadecimal. Note que neste caso a compatibilidade entre o Simus e o Neanderwin não é mantida, ou seja, a imagem salva em um simulador não pode ser carregada em outro.

A Figura 3 mostra a aparência da tela principal do simulador SimuS. Na parte superior estão o menu geral de operação (Arquivo, Editar, etc.) e diversos botões usados em conjunto com o editor de textos, que seguem o estilo usual de programas com interface gráfica.

Uma vez criado o programa ele será compilado, o que provoca o aparecimento de uma janela pop-up com a listagem, cujo formato de saída é similar à maioria dos montadores profissionais, com indicação dos eventuais erros de compilação. O programa compilado é carregado imediatamente

na memória, sendo o conteúdo atualizado e exibido no painel correspondente. Caso se deseje, é possível copiar o conteúdo desta janela para a área de transferência, para colar em algum editor de textos possibilitando eventuais embelezamentos e impressão a posteriori.

O número de instruções do processador Sapiens é relativamente pequeno, mas apesar disso notamos que é importante tornar disponível uma "ajuda online" interativa, que é ativada pelo menu do programa, um sistema de entrada interativa de instruções e pseudo-instruções. A função de criação tutorada de programas faz com que o aluno cometa menos erros e a compreenda melhor o significado das instruções e produza uma sintaxe das instruções mais correta, de forma interativa.

Por último, notamos que para muitos estudantes o domínio das representações hexadecimal e binária, necessário para verificar e alterar a memória, só se dá depois de algum tempo. Mesmo sabendo que as calculadoras do Windows e do Linux exibem resultados em várias bases, incluímos um conversor de bases, que é muito mais simples.

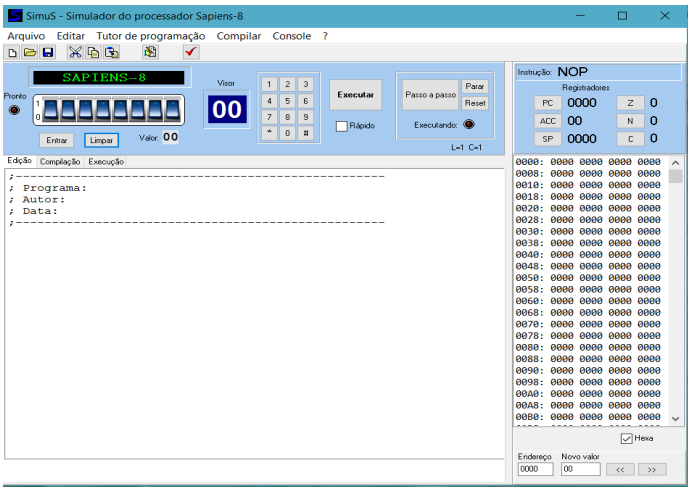

Fig. 3. Tela Principal do Simulador Simus

REFERÊNCIAS

[1] J. A. S. Borges, G. P. Silva "NeanderWin - um simulador didático para uma arquitetura do tipo acumulador". WEAC, 2006.

[2] R. F. Weber, Fundamentos de Arquitetura de Computadores. 2. Ed. Porto Alegre: Instituto de Informática da UFRGS: Sagra Luzzatto, 2001.

[3] Erick V. C. L. Borges et alli, "SEAC: um simulador online para ensino de arquitetura de computadores" - WEAC, 2012