Linguagem Assembler para o Z01

Rafael Corsi @ 2018, Luciano Pereira @ 2017

A linguagem do Assembly usada pelo Z01 é baseada na sintaxe AT&T (usada na implementação do Assembler GNU gas). Essa sintaxe possui diversas diferenças da sintaxe Intel.

Formato das Instruções
Em Assembly, codificamos uma instrução por linha e ela é construída a partir do mnemônico da operação e seus argumentos. Além disso, temos marcadores de posição (endereço usado em desvios), constantes e possíveis variáveis. O formato das instruções na sintaxe AT&T segue o seguinte formato:

mnemônico origem, destino

Registradores
Todos os registradores devem ter como prefixo o sinal de porcentagem '%', por exemplo: %A ou %D ou %S.

Os registradores %S e %D compartilham a mesma entrada da ALU, impossibilitando que sejam utilizados simultaneamente, por exemplo :
~~addw %D, %S, %A~~

Valores Literais
Todos os valores literais devem ter como prefixo o sinal de cifrão '$', por exemplo: $55, $376, sendo o maior valor 2047 (15 bits).

Endereçamento de Memória
Na sintaxe AT&T, a memória é referenciada com parêntese em volta do registrador que armazena o endereço: por exemplo (%A).

Tamanho dos operadores
Algumas instruções podem trabalhar com diferentes tamanhos de dados, assim as instruções podem ter um sufixo, informando o tamanho do dado que irá manipular, sendo b (8 bits), w (16 bits) e l (32 bits). Por exemplo:
movw $2000, (%A)

Instruções de Transferência de Controle
As instruções de jump, fazem o fluxo do programa desviar de uma posição do programa para outra. Para marcar as posições no programa, são usados marcadores (labels) que sempre terminam com dois pontos (:). Por exemplo: loop:

Registradores virtuais
Os símbolos R0, ..., R15 são automaticamente predefinidos para se referir aos endereços de RAM 0, ..., 15

Ponteiros de I/O
Os símbolos SCREEN e KBD são automaticamente predefinidos para se referir aos endereços de RAM 16384 e 24576, respectivamente.

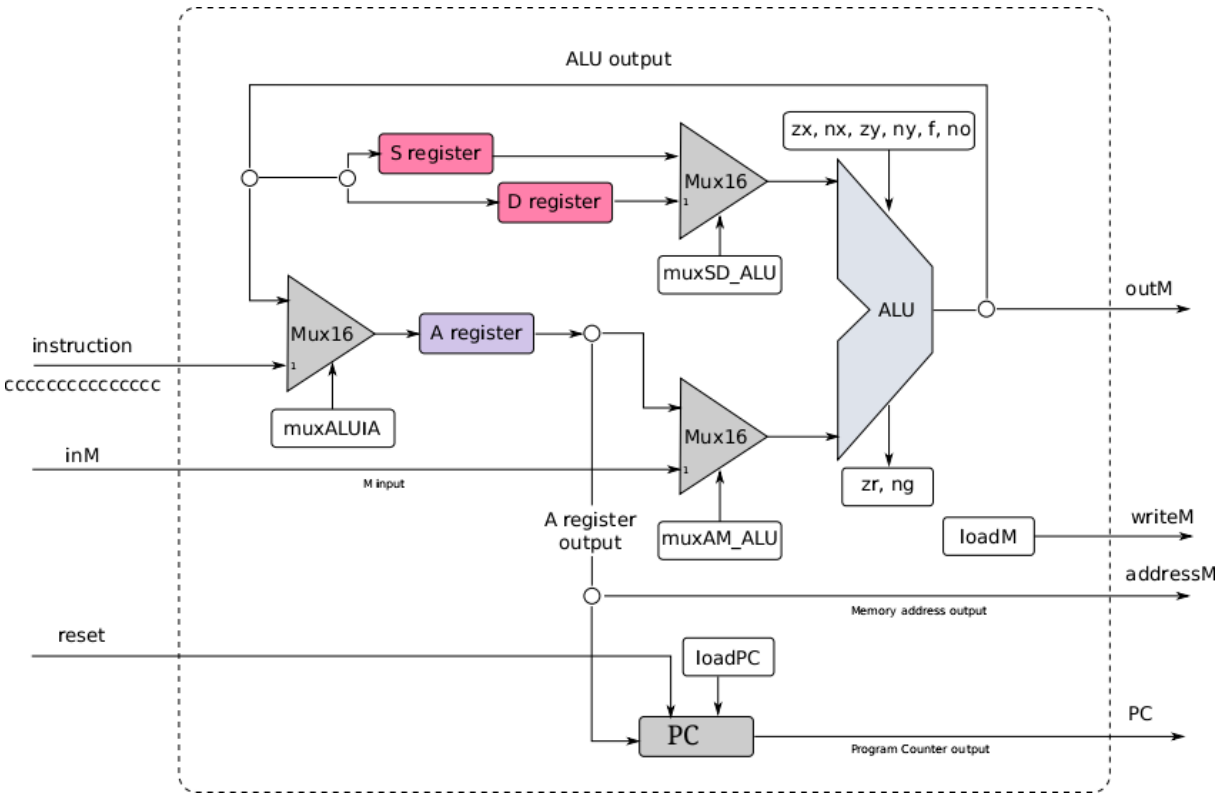
Ponteiros de controle da VM
Os símbolos SP, LCL, ARG, THIS, e THAT são automaticamente predefinidos para se referir aos endereços de RAM 0-4, respectivamente.

Notações:
im : valor imediato (somente os valores 1, 0 e -1).
reg : registrador.
mem: memória, ou seja (%A).

Limitações:
- A arquitetura não permite somar o valor da memória apontada por (%A) com o valor de %A, ou de (%A) com (%A), tampouco %A com %A.
- Não é possível somar (ou subtrair, se é que isso faz sentido) o registrador com o mesmo, por exemplo somar %D com %D.
- Não é possível ler e gravar a memória ao mesmo tempo. Por exemplo, as instruções abaixo não funcionam no nosso computador:
    - incw (%A);
    - subw (%A),%D,(%A).

Observação:
A linguagem Assembly apresentada é especifica para o processador produzido no curso. Embora muito similar a outras usadas em produtos de mercado, as instruções possuem limitações inerentes a cada hardware.

CPU
A CPU proposta para o desenvolvimento do computador possui a arquitetura ilustrada a seguir.



Instruções Assembly

LEA - Carregamento Efetivo do Endereço (Valor)

leaw im[15], reg[16]

Descrição: A instrução *lea* armazena o valor passado (im[15]) no registrador especificado (na nossa implementação, somente o %A).

Exemplo : *A = 15*

Assembly: leaw $15,%A

MOV - Copia Valores

movw im*/reg/mem, reg/mem {, reg/mem, reg/mem}

***(o imediato pode utilizar somente os valores: 1, 0 ou -1)**

Descrição: A instrução *mov,* copia o valor da primeira posição para a segunda posição (terceira e quarta opcional)**.

****(caso sejam utilizadas, serão destinos adicionais para o valor da primeira posição).**

Exemplo : S = *RAM[A]*

Assembly: movw (%A),%S

ADD - Adição de Inteiros

addw reg/mem, reg/mem/im*, reg/mem {, reg/mem, reg/mem}

Descrição: instrução r*sub,* subtrai o primeiro valor do segundo valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional)**.

****(caso sejam utilizadas, serão destinos adicionais para o resultado da operação).**

Exemplo : S = *RAM[A]+D*

Assembly: addw (%A), %D, %S

SUB - Subtração de Inteiros

subw reg/mem, rem/mem/im*, reg/mem {, reg/mem, reg/mem}

Descrição: A instrução *sub,* subtrai o segundo valor do primeiro valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional)**.

****(caso sejam utilizadas, serão destinos adicionais para o resultado da operação).**

Exemplo : *A = D-RAM[A]*

Assembly: subw %D, (%A), %A

RSUB - Subtração de Inteiros Reversa

rsubw reg/mem/im*, rem/mem, reg/mem {, reg/mem, reg/mem}

Descrição: A instrução r*sub,* subtrai o segundo valor do primeiro valor e armazena o resultado no terceiro parâmetro (quarto e quinto opcional)**.

****(caso sejam utilizadas, serão destinos adicionais para o resultado da operação).**

Exemplo : *A = RAM[A]-D*

Assembly: rsubw %D, (%A), %A

INC - Incrementa Inteiro

incw reg/mem

Descrição: A instrução *inc,* adiciona um (1) ao valor do registrador ou memória.

Exemplo : *D = D+1*

Assembly: incw %D

DEC - Decrementa Inteiro

decw reg/mem

Descrição: A instrução *dec,* subtrai um (1) do valor do registrador ou memória.

Exemplo : *A = A-1*

Assembly: decw %A

NOT – Negação por Complemento de Um

notw reg/mem

Descrição: A instrução *not,* inverte o valor de cada bit do argumento, ou seja, se um bit tem valor 0 fica com 1 e vice-versa.

Exemplo : *D = !D*
Assembly: notw %D

NEG – Negação por Complemento de Dois

negw reg/mem

Descrição: A instrução *neg,* faz o valor ficar negativo, ou seja, um valor de x é modificado para -x.

Exemplo: *A = -A*
Assembly: negw %A

AND – Operador E (and)

andw reg/mem, rem/mem

Descrição: A instrução *and* executa o operador lógico E (and).

Exemplo : ***D = A&D***

Assembly: andw %A, %D, %D

OR – Operador OU (or)

orw reg/mem, rem/mem

Descrição: A instrução *or* executa o operador lógico Ou (or).

Exemplo: *D = RAM[A]|D*

Assembly: orw (%A), %D, %D

JMP – Jump

jmp

Descrição: A instrução *jmp* executa um desvio, no fluxo de execução, para o endereço armazenado em %A.

Assembly: jmp

JE – Desvia Execução se Igual a Zero

je reg

Descrição: A instrução *je* faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for igual a zero.

Assembly: je %S

JNE – Desvia Execução se Diferente de Zero

Jne reg

Descrição: A instrução *jne* faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for diferente de zero.

Assembly: jne %D

JG – Desvia Execução se Maior que Zero

jg reg

Descrição: A instrução *jg* desvia, o fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for maior que zero.

Assembly: jg %S

JGE – Desvia Execução se Maior Igual a Zero

jge reg

Descrição: A instrução *jge* faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for maior ou igual a zero.

Assembly: jge %S

JL – Desvia Execução se Menor que Zero

jl reg

Descrição: A instrução *jl* faz desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for menor que zero.

Assembly: jl %S

JLE – Desvia Execução se Menor Igual a Zero

jle reg

Descrição: A instrução *jle* faz um desvio, no fluxo de execução, para o endereço armazenado em %A, somente se o valor do reg. for menor ou igual a zero.

Assembly: jle %D

NOP – Não faz nada (No Operation)

nop

Descrição: A instrução *nop* não faz nada, usado para pular um ciclo de execução.

Assembly: nop
