



Universidade Federal de Pelotas
 Instituto de Física e Matemática
 Departamento de Informática
 Bacharelado em Ciência da Computação

Introdução à Ciência da Computação

Conjunto de Instruções, Formato de Instruções e Número de Endereços

Prof. Gerson Cavalheiro

<http://www.ufpel.edu.br/~gerson.cavalheiro>

<http://gersonc.arahy.org>

(Material original: Prof. José A. Gunzital)

Programa: definição de alto nível

Um programa é uma seqüência finita de comandos (em alguma linguagem de programação), com o propósito de resolver um problema específico. Exemplo:

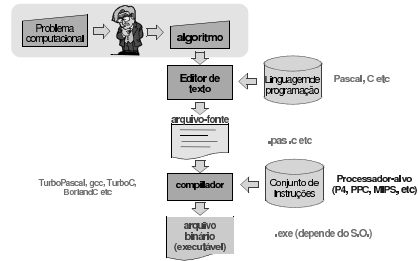
```
#include <stdio.h>
int main(int rv, int cont)
{
    int total=0;
    while( cont > 0 ){
        total += rv;
        cont--;
    }
    return total;
}
```

Programa: definição de baixo nível (de máquina)

Um programa é uma seqüência finita de instruções (de um dado processador), com o propósito de resolver um problema específico. Exemplo:

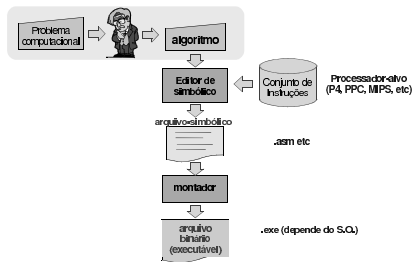
```
LDR A #0
LDR X 129
LDR B 128
JZ 16
ADD A,0,X
ADD X,#1
SUBB #1
JMP 6
STR A,130
HLT
```

Fluxo de Desenvolvimento de um Programa



Fluxo de Desenvolvimento de um Programa - 2

Nível de abstração baixo



Instrução

Uma instrução é uma operação básica que um processador (ou um modelo/família de processadores) é capaz de realizar.

Exemplos de instruções (e seus significados):

instrução	significado
NOP	Nenhuma operação
STA end	Armazena acumulador (store)
LDA end	Carrega acumulador (load)
ADD end	Soma
OR end	"OU" lógico
AND end	"E" lógico
NOT	Inverte (complementa) acumulador
JMP end	Desvio incondicional (JMP)
JN end	Desvio condicional (jump on negative)
JZ end	Desvio condicional (jump on zero)
HLT	Término de execução (halt)

► Formato de Instrução

- Cada instrução é representada como um conjunto de bits
- Os bits são agrupados em campos, a fim de facilitar o projeto do bloco de controle
- Os campos definem o formato da instrução. Exemplo de formato:



► Conjunto de Instruções

Cada processador (modelo ou família de processador) possui um conjunto de instruções próprio.

instrução	significado
NOP	Nenhuma operação
STA end	Armazena acumulador (store)
LDA end	Carrega acumulador (load)
ADD end	Soma
OR end	"OU" lógico
AND end	"E" lógico
NOT	Inverte (complementa) acumulador
JMP end	Desvio incondicional (jump)
JN end	Desvio condicional (jump on negative)
JZ end	Desvio condicional (jump on zero)
HLT	Término de execução (halt)

► Conjunto de Instruções

Instrução definida como uma transferência de dados entre registradores (e/ou memória), com ou sem modificação.

Exemplos:

instrução	transferência
NOP	Nenhuma operação
STA end	MEM(end) ← AC
LDA end	AC ← MEM(end)
ADD end	AC ← MEM(end) + AC
OR end	AC ← MEM(end) OR AC
AND end	AC ← MEM(end) AND AC
NOT	AC ← NOT AC
JMP end	PC ← end
JN end	IF N=1 THEN PC ← end
JZ end	IF Z=1 THEN PC ← end

► Conjunto de Instruções

Tipos de instruções:

- Instruções de transferência de dados
 - load, store, move
- Instruções aritméticas e lógicas
 - add, sub, mul, div, and, or, xor, not
- Instruções de teste e de desvio (ou de controle de execução)
 - jump, jump on negative (JN), jump on zero (JZ), branch on equal (beq)

► Definições

- Muitas instruções realizam operações sobre operandos. Exemplos: instruções aritméticas e lógicas
- Os operandos podem estar em qualquer posição da memória ou em qualquer registrador (entre os de propósito geral)
- Assim, instruções que definem uma operação sobre um operando precisam conter também o endereço do operando
- No caso de mais de um operando, cada operando pode ter um endereço diferente
- No caso de instruções de desvio, a instrução deve indicar para qual posição ou endereço de programa se quer desviar

► Endereçamento

- Operandos podem estar na memória ou em algum registrador
- Operandos que estão na memória são endereçados por um (ou mais) endereços de memória
- Operandos que estão nos registradores são endereçados por um ou mais endereços de registrador
- Os computadores podem ter instruções de diferentes comprimentos (números de bits), com um número de endereços que pode ir de 0 a 4 (tipicamente)
- O número de campos de endereço no formato de instrução depende da organização interna do processador e do seu número de registradores
- Tipos de organização:
 - Organização com acumulador
 - Organização com múltiplos registradores
 - Organização de pilha

► Endereçamento de Memória

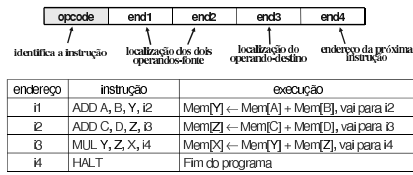
- Os computadores podem ter instruções de diferentes comprimentos (números de bits), com um número de endereços que pode ir de 0 a 4 (tipicamente)
- Para analisar o impacto do número de campos de endereço de memória e do número de operandos na memória, consideremos que se deseje programar a seguinte equação:

$$X = (A+B) * (C+D)$$

onde X, A, B, C e D são endereços de memória.

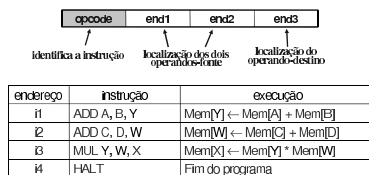
- Consideremos também que tenhamos as seguintes instruções disponíveis: ADD, SUB, MUL, MOVE (ou LOAD e STORE) e HALT (fim de programa)

► Formato de Instrução com 4 Endereços



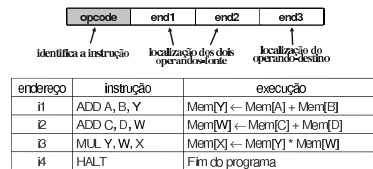
- Vantagem:** não há necessidade de instruções de desvio do fluxo de execução (jump, branch)
- Desvantagem:** os programas são sequenciais, i.e., na maior parte das vezes, a próxima instrução está no próximo endereço de memória

► Formato de Instrução com 3 Endereços



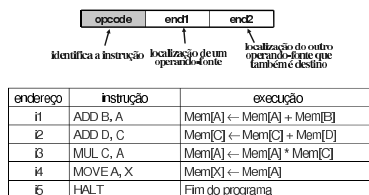
- Usa um registrador apontador de programa (PC), o qual é incrementado para apontar para a próxima instrução a ser executada (exceto nos desvios)
- No conjunto de instruções deve haver instruções de desvio do fluxo de execução (jump, branch)

► Formato de Instrução com 3 Endereços



- Y e W são endereços de memória, usados para armazenamento temporário
- São feitos 9 acessos à memória. Quantos acessos seriam feitos à memória se o campo "end3" fosse um endereço de registrador?

► Formato de Instrução com 2 Endereços



- A e C são sobrescritos!
- São feitos 11 acessos à memória. Quantos acessos seriam feitos à memória se o campo fosse possível usar registradores para armazenamento temp.?

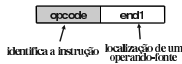
► Formato de Instrução com 2 Endereços

Usando dois registradores para armazenamento temporário (R1 e R2)

endereço	instrução	execução
i1	MOVE A, R1	R1 ← Mem[A]
i2	ADD B, R1	R1 ← R1 + Mem[B]
i3	MOVE C, R2	R2 ← Mem[C]
i4	ADD D, R2	R2 ← R2 + Mem[D]
i5	MUL R2, R1	R1 ← R1 * R2
i6	MOVE R1, X	Mem[X] ← R1
i7	HALT	Fim do programa

- O programa tem mais instruções.
- Porém, são feitos apenas 5 acessos à memória.

► Formato de Instrução com 1 Endereço



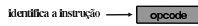
- Há um registrador chamado acumulador (abreviado por AC), que contém o segundo operando
- O acumulador também é o destino do resultado
- As instruções de movimentação de dados são LOAD e STORE, as quais são definidas em relação ao acumulador (AC).

► Formato de Instrução com 1 Endereço

endereço	instrução	execução
i1	LOAD A	$AC \leftarrow Mem[A]$
i2	ADD B	$AC \leftarrow AC + Mem[B]$
i3	STORE Y	$Mem[Y] \leftarrow AC$
i4	LOAD C	$AC \leftarrow Mem[C]$
i5	ADD D	$AC \leftarrow AC + Mem[D]$
i6	MUL Y	$AC \leftarrow AC * Mem[Y]$
i7	STORE X	$Mem[X] \leftarrow AC$
i8	HALT	Fim do programa

- Usa uma posição de memória (Y) para armazenamento temporário!
- São feitos 7 acessos à memória.

► Formato de Instrução com 0 Endereço



- Não existe referência sobre a localização dos operandos (memória ou registrador)
- A localização dos operandos se dá de maneira implícita, quando a arquitetura foi definida
- Uma possibilidade é colocar os operandos em uma área específica da memória e fazer uso de um mecanismo específico para alocação.

► Formato de Instrução com 0 Endereço

- Exemplo; uma pilha
 - os operandos são sempre retirados do topo da pilha
 - o resultado é colocado no topo da pilha
- Para facilitar o uso da pilha, as equações são expressas usando a notação polonesa reversa (o símbolo da operação é escrito após os operandos):
 - Use de duas instruções para acessar os operandos:
 - $A B C D + + *$
- PUSH (coloca no topo da pilha)
- POP (retira do topo da pilha)

► Formato de Instrução com 0 Endereço

endereço	instrução	execução
i1	PUSH A	$topo \leftarrow Mem[A]$
i2	PUSH B	$topo \leftarrow Mem[B]$
i3	ADD	$topo \leftarrow (Mem[A] + Mem[B])$ (A e B são retirados do topo da pilha)
i4	POP X	$Mem[X] \leftarrow topo$
i5	PUSH C	$topo \leftarrow Mem[C]$
i6	PUSH D	$topo \leftarrow Mem[D]$
i7	ADD	$topo \leftarrow (Mem[C] + Mem[D])$ (C e D são retirados do topo da pilha)
i7	PUSH X	$topo \leftarrow Mem[X]$
i8	MUL	$topo \leftarrow ((Mem[C] + Mem[D]) * (Mem[C] + Mem[D]))$
i9	POP A	$Mem[A] \leftarrow topo$
i10	HALT	Fim do programa

► Neander

Computador hipotético Neander

- Largura de dados e endereços de 8 bits
- Dados representados em complemento de 2
- 1 acumulador de 8 bits (AC)
- 1 apontador de programa de 8 bits (PC)
- 1 registrador de estado com 2 códigos de condição: negativo (N) e zero (Z)
- Memória de 256 posições (programa e dados):
 - Área de programa
 - posição 0 (0H): início do programa
 - Área de dados
 - posição 128 (80H): início da área de dados

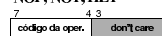
► A Arquitetura: conjunto de instruções

instrução	comentário	transferência
NOP	Nenhuma operação	Nenhuma operação
STA end	Armazena acumulador (store)	MEM(end) ← AC
LDA end	Carrega acumulador (load)	AC ← MEM(end)
ADD end	Soma	AC ← MEM(end) + AC
OR end	"OU" lógico	AC ← MEM(end) OR AC
AND end	"E" lógico	AC ← MEM(end) AND AC
NOT	Inverte (complementa) acumulador	AC ← NOT AC
JMP end	Desvio incondicional (jump)	PC ← end
JN end	Desvio condicional (jump on negative)	IF N=1 THEN PC ← end
JZ end	Desvio condicional (jump on zero)	IF Z=1 THEN PC ← end

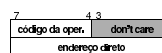
► A Arquitetura: códigos das operações (opcodes)

instrução	Código
NOP	0000
STA end	0001
LDA end	0010
ADD end	0011
OR end	0100
AND end	0101
NOT	0110
JMP end	1000
JN end	1001
JZ end	1010
HLT	1111

Instruções com um byte:
NOP, NOT, HLT

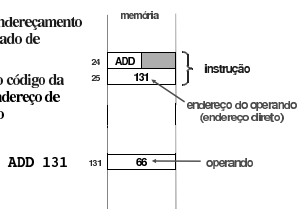


Instruções com dois bytes:
STA, LDA, ADD, OR, AND,
JMP, JN, JZ



► A Arquitetura: modos de endereçamento

- Somente o modo de endereçamento direto (também chamado de absoluto)
- A palavra que segue o código da instrução contém o endereço de memória do operando
- Exemplo:



► A Arquitetura: códigos de condição

A ULA do Neander fornece os seguintes códigos de condição, os quais são usados pelas instruções JN e JZ

N (negativo): sinal do resultado de uma operação na ULA

- se o resultado da ULA for negativo, N=1
- Caso contrário, N=0

Z (zero): indica resultado igual a zero

- Se o resultado da ULA for zero, Z=1
- Caso contrário, Z=0

► A Arquitetura: programação

- Programa e os dados estarão armazenados na memória
- Deve ser escolhida uma área de programa e uma área de dados
- A área de programa não deve invadir a área de dados e vice-versa
- Por convenção a área de programa ocupa a metade inferior dos endereços e a área de dados ocupa a metade superior
- Aliás, qual é o tamanho de memória que o Neander consegue endereçar?

► A Arquitetura: programação

- O Neander usa 8 bits para endereçar (= largura de endereço de 8 bits)
- Logo, ele consegue acessar qualquer endereço do intervalo:
 - 00000000 a 11111111 (em binário)
 - 0 a 255 (em decimal)
 - 0H a FFH (em hexadecimal)
- Então, iremos adotar a seguinte divisão da memória do Neander:
 - Área de programa: posição 0H até 7FH
 - Área de dados: posição 80H até FFH

► A Arquitetura: programação

- **Exemplo:** Programa que soma o conteúdo de 3 posições consecutivas da memória e armazena o resultado na quarta posição.

Configuração da memória:

Área de programa
posição 0 (0H): início do programa

Área de dados
posição 128 (80H): primeira parcela
posição 129 (81H): segunda parcela
posição 130 (82H): terceira parcela
posição 131 (83H): resultado

simbólico	comentário	decimal	hexa
LDA 128	AC ← MEM(128)	32 128	20 80
ADD 129	AC ← AC + MEM(129)	48 129	30 81
ADD 130	AC ← AC + MEM(130)	48 130	30 82
STA 131	MEM(131) ← AC	16 131	10 83
HLT	Término da execução	240	FD

Programa

Representação interna

► Neander: Exercícios

- Baixar Neander e desenvolver os programas abaixo. Documentar cada programa e entregar na aula do dia 14/junho.

Fazer um programa que:

1. some dois valores que se encontram em posições contíguas de memória e armazene o resultado em uma terceira posição (por exemplo, somar 80H e 81H e colocar o resultado em 82H).
2. multiplique por 3 um valor na posição de memória 80H e coloque o resultado na posição de memória 81H
3. troque o conteúdo de duas posições de memória contíguas.
4. coloque em na posição de memória 81H o conteúdo do valor armazenado na posição 80H.
5. decrimenta de 1 um valor armazenado na posição de memória 80H
6. encontre o maior número de dois valores em duas posições contíguas de memória e coloque este valor em uma terceira posição de memória.
7. faça o somatório de todos os números no intervalo entre 0 e o conteúdo da posição de memória 80H.
7. calcule quanto vale o conteúdo da posição de memória 80H elevado ao conteúdo da posição de memória 81H. Coloque o resultado em 80H e 81H.