Arquitetura de Computadores

Codificação de instruções

Compromissos na especificação de operandos no ISA do P16
Bib: A – secção 6.4 e B – secção A.7

Slides inspirados nos slides do prof. Tiago Dias

João Pedro Patriarca (jpatri@cc.isel.ipl.pt), Gabinete F.0.23 do edifício F

ISEL, ADEETC, LEIC

Sumário

- Codificação de instruções
  - Considerações gerais
  - P16

- Compromissos na especificação de operandos no ISA do P16
  - Acessos a memória
    - Baseado no registo PC
    - Baseado e indexado com constante
  - Instruções de salto
  - Constantes de 16 bits
  - Processamento de dados

Codificação de instruções
Considerações gerais

- Componentes que constituem a codificação de uma instrução
  - Código de operação (*operation code* ou *opcode*): identifica a instrução permitindo definir as ações que são realizadas na execução da instrução
  - Código dos operandos: operandos envolvidos na execução da instrução
- O código de uma instrução obedece a um formato (número e organização dos campos que constituem a instrução)
  - A sistematização da organização de campos do mesmo tipo tem implicações na complexidade do hardware que depende desses campos
- A dimensão de cada campo está dependente das características do conjunto de instruções
  - A dimensão do *opcode* está relacionada com o número de instruções suportadas
  - A dimensão dos operandos depende do número de operandos do mesmo tipo e do número de bits disponíveis para a sua codificação

Codificação de instruções
Considerações gerais – classes de códigos

Comprimento variável	
Vantagens	- Cada instrução com a dimensão necessária para codificar o *opcode* e os seus operandos - Maior versatilidade na especificação de operandos: por exemplo, definição de endereços diretos com capacidade de endereçar na totalidade o espaço de endereçamento ou, por exemplo, a definição de constantes com dimensão suficiente para especificar valores completos de registos - Facilidade em associar várias ações na execução de cada instrução: produzem programas com menor número de instruções
Desvantagens	- Maior complexidade do hardware de suporte ao carregamento de instruções de memória e descodificação de instruções - Pior desempenho na execução das instruções: mais tempo para carregar uma instrução de memória; mais tempo na descodificação de uma instrução; tempos de execução díspares entre instruções
Comprimento fixo	
Vantagens	- Tipicamente, o carregamento do código de uma instrução é feito num único acesso a memória - Formato de instruções uniforme simplificando o hardware e tornando-o mais eficiente - Facilita a utilização em paralelo de hardware com responsabilidades distintas (arquiteturas em Pipeline)
Desvantagens	- Maior compromisso na codificação de operandos - Maior desperdício de bits (bits não usados) - Utilização de um maior número de instruções na produção de um programa porque cada instrução tem associada, tipicamente, apenas uma ação

Codificação de instruções
P16 – características principais da arquitetura

- Processador de 16 bits baseado na arquitetura ARMv4
- Características comuns com as arquiteturas *Reduced Instruction Set Computer* (RISC)
  - Arquitetura *Load/Store*
  - Banco de registos genérico: 16 registos
  - Operações da ALU a 16 bits
  - Conjunto de instruções reduzido (43 instruções)
    - Organizadas em três classes: manipulação de dados, transferência de dados e controlo
    - Suporte à implementação de rotinas
    - Suporte à implementação de estrutura de dados pilha (*stack*)
    - Modos de endereçamento simples: baseado/indexado e relativo
- Espaço de endereçamento de 64 KB
  - Espaço único para código, dados e acesso a periféricos
  - Acessível a 8 e a 16 bits
- Suporte para o processamento de interrupções externas

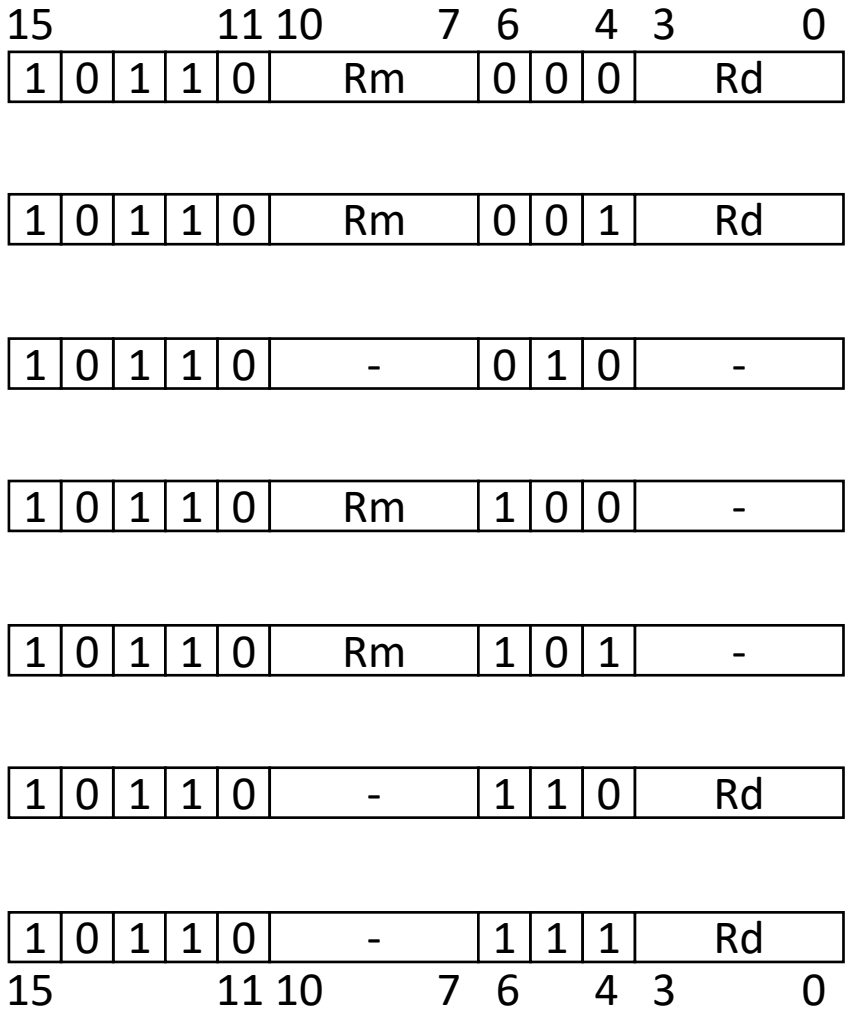
Codificação de instruções
P16 – características principais das instruções

- Dimensão fixa: 16 bits
- Sem suporte de endereçamento direto
- Processamento de dados baseado exclusivamente em registos
- Cinco formatos de codificação
  - Cópia de dados
  - Processamento de dados
  - Controlo
  - Memória
  - Carregamento de constantes

- As instruções de processamento de dados, controlo, memória e carregamento de constantes foram já apresentadas em sessões anteriores

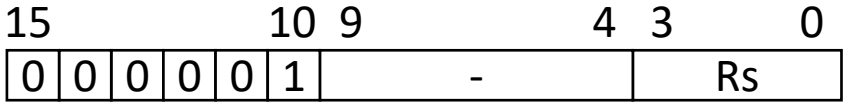
Codificação de instruções
P16 – instruções de cópia de dados

```
MOV Rd, Rm ; Rd=Rm

MVN Rd, Rm ; Rd=~Rm

MOVS PC, LR ; PC=LR e CPSR=SPSR

MSR CPSR, Rm ; CPSR=Rm

MSR SPSR, Rm ; SPSR=Rm

MRS Rd, CPSR ; Rd=CPSR

MRS Rd, SPSR ; Rd=SPSR
```

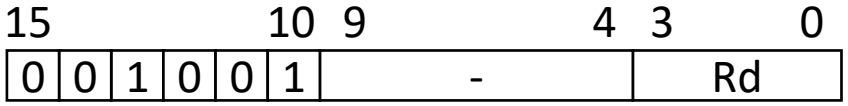


Codificação de instruções
P16 – instruções de acesso a memória

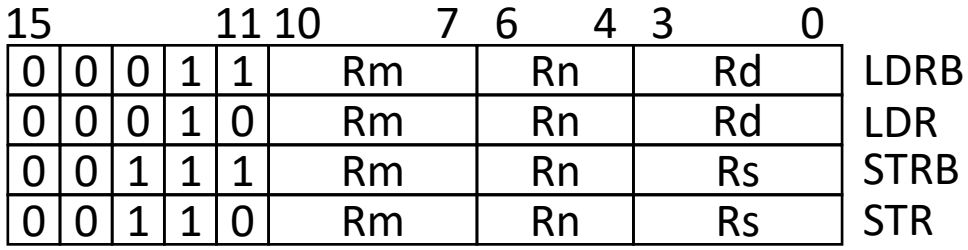
```
PUSH Rs ; SP-=2 depois mem[SP]=Rs
```




```
POP Rd  ; Rd=mem[SP] depois SP+=2
```




```
LDRB Rd, [Rn, #imm3] ; Rd8-15=0, Rd0-7=mem[Rn+imm3]
LDRB Rd, [Rn, Rm]    ; Rd8-15=0, Rd0-7=mem[Rn+Rm]

LDR  Rd, [Rn, #imm4] ; Rd=mem[Rn+imm4]
LDR  Rd, [Rn, Rm]    ; Rd=mem[Rn+Rm]

STRB Rs, [Rn, #imm3] ; mem[Rn+imm3]=Rs0-7
STRB Rs, [Rn, Rm]    ; mem[Rn+Rm]=Rs0-7

STR  Rs, [Rn, #imm4] ; mem[Rn+imm4]=Rs
STR  Rs, [Rn, Rm]    ; mem[Rn+Rm]=Rs
```

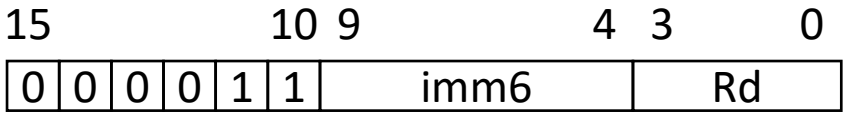


Sumário

- Codificação de instruções
  - Considerações gerais
  - P16

- Compromissos na especificação de operandos no ISA do P16
  - Acessos a memória
    - Baseado no registo PC
    - Baseado e indexado com constante
  - Instruções de salto
  - Constantes de 16 bits
  - Processamento de dados

Acessos a memória
Baseado no registo PC (relativo)



`LDR Rd, labelS ; Rd = mem[PC + #imm6*2]`

- A constante `imm6` é um valor expresso no domínio dos naturais baseado na diferença entre o endereço de memória associado a `labelS` e o valor atual do registo PC durante a execução da instrução LDR
- A posição de memória a aceder não pode ter um endereço inferior ao valor do registo PC
- A diferença entre a posição de memória a aceder e o valor atual do registo PC não pode ser superior a 64 *words*

```
   .data
a: .word    0x1234
b: .byte    0x56

   .text
   ldr  r0, a ; erro: endereço de a
              ; inferior a PC
   ldrb r1, b ; erro: não inclui versão
              ; para acesso ao byte
```

```
   .data
a: .word   0x1234
b: .byte   0x56

   .text
   ldr   r0, a_addr
   ldr   r0, [r0]
```

```
   ldr   r1, b_addr
   ldrb  r1, [r1]
   ; ...
a_addr:  .word   a
b_addr:  .word   b
```

Acessos a memória
Baseado e indexado com constante

	15					10	9 7	6 4	3 0
LDRB	0	0	0	0	1	0	imm3	Rn	Rd
STRB	0	0	1	0	1	0	imm3	Rn	Rs
LDR	0	0	0	0	0	0	$imm4_{3-1}$	Rn	Rd
STR	0	0	1	0	0	0	$imm4_{3-1}$	Rn	Rs

```
LDRB/STRB Rd/s, [Rn, #imm3] ; Rd=mem[Rn+imm3]
                            ; mem[Rn+imm3]=Rs
```

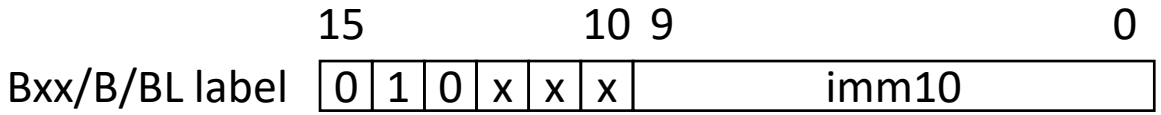
```
LDR/STR Rd/s, [Rn, #imm4] ; Rd=mem[Rn+imm4]
                          ; mem[Rn+imm4]=Rs
```

- Na codificação da instrução `LDRB/STRB`, a constante `imm3` expressa a distância em *bytes* em relação a `Rn`
- Na codificação da instrução `LDR/STR`, a constante `imm4` expressa a distância em *words* em relação a `Rn`
- No *assembly*, `imm3/imm4` expressa sempre a distância em *bytes*, quer para acesso a *byte*, quer para acesso a *word*

```
   .data
a: .space 7
   .byte 0x34, 0x56, 0x78, 0x9a
   .text
   ldr   r0, addr_a
   ldrb  r1, [r0, #7]  ; r1 = 0x34
   ldrb  r1, [r0, #8]  ; erro de codificação
   mov   r2, #8
   ldrb  r1, [r0, r2]  ; r1 = 0x56
   ...
addr_a:  .word  a
```

```
   .data
a: .space 14
   .word 0x34, 0x56, 0x78, 0x9a
   .text
   ldr   r0, addr_a
   ldr   r1, [r0, #14] ; r1 = 0x0034
   ldr   r1, [r0, #16] ; erro de codificação
   mov   r2, #16
   ldr   r1, [r0, r2]  ; r1 = 0x0056
   ...
addr_a:  .word  a
```

Instruções de salto

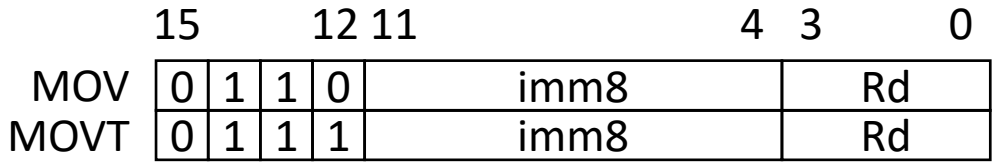


- Em caso de salto faz: `PC = PC + imm10*2`
- A constante `imm10` é um valor expresso no domínio dos relativos: `imm10` ∈ [-512, +511] *words*
- A constante codificada expressa a diferença em *words* entre o valor atual do PC durante a execução da instrução de salto e o endereço de memória associado a `label`
- No *assembly*, o caráter '.' representa salto para a própria instrução e é traduzido pela constante -1

```
   .text
   b      l0
   .space 1022
l0:
   b      l1 ; erro: diferença não
             ; codificável com 10 bits
   .space 1026
l1:
   b      .
```

```
   .text
   b      l0
   .space 1022
l0:
   ldr    pc, addr_l1
addr_l1:
   .word  l1
   .space 1026
l1:
   b      .
```

Constantes de 16 bits




```
MOV Rd, #imm8    ; Rd0-7 = imm8, Rd8-15 = 0
MOVT Rd, #imm8   ; Rd8-15 = imm8, Rd0-7 = Rd0-7
LDR Rd, labelS   ; Rd = mem[PC + imm6*2]
```

```
   .text
   mov   r0, #0x1234; erro: constante
                     ; não codificável
                     ; com 8 bits
   b     .
```

```
   .text
   ; opção 1
   mov    r0, #0x34 ; r0 = 0x0034
   movt   r0, #0x12 ; r0 = 0x1234
   b      .
   ; opção 2
   ldr    r0, Const1234
   b      .
Const1234:
   .word  0x1234
```

Processamento de dados

	15			11	10 7	6 4	3 0
ALU Rd, Rn, Rm	1	x	x	x	x	Rm	Rn	Rd
ALU Rd, Rn, imm4	1	x	x	x	x	imm4	Rn	Rd
CMP Rn, Rm	1	0	1	1	1	Rm	Rn	-
RRX Rd, Rn	1	1	0	1	1	-	Rn	Rd

```
ALU Rd, Rn, Rm    ; Rd = Rn AL Rm
ALU Rd, Rn, #imm4 ; Rd = Rn AL imm4
CMP Rn, Rm        ; Rn – Rm
RRX Rd, Rn        ; Rd = Rn >> 1, Rd15 = Cy, Cy = Rn0
```

- Rd ∈ [R0..R15]
- Rn ∈ [R0..R7]
- Rm ∈ [R0..R15]
- imm4 ∈ [0..15]

- Estratégias em cenários onde Rn >= R8:
  1. Verificar se é viável e adequado inverter os operandos
     - Se Rm original corresponde a um registo < R8
     - Se numa comparação, a inversão não implica mais do que uma instrução de salto
  2. Usar temporariamente para Rn um registo < R8

QUIZ 4 – ISA do P16

Access code: A8YEN