Conjuntos de instruções de microprocessadores

Arquitetura ARMv7

João Canas Ferreira

Novembro de 2018

U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Tópicos

1. Arquitetura do conjunto de instruções
2. Codificação de instruções ARMv7
3. Programação em Assembly
4. Definição e utilização de sub-rotinas

Contém figuras de "Computer Organization and Design", D. Patterson & J. Hennessey, 3ª. ed., MKP

1 Arquitetura do conjunto de instruções

2 Codificação de instruções ARMv7

3 Programação em Assembly

4 Definição e utilização de sub-rotinas

Dois princípios

➠ Os computadores atuais seguem dois princípios-chave:

1. Instruções são representadas como números;
2. Programas (sequências de instruções) são guardados em memória, tal como dados.

➠ Programas podem ser fornecidos como ficheiros (de dados binários): os dados são as instruções do programa.

➠ Esses programas podem ser executados em computadores que aceitem o mesmo conjunto de instruções codificadas da mesma maneira: *compatibilidade binária*.

➠ Um programa (A) também pode ser executado por outro programa (V), que *interpreta* as instruções de A: V é um *simulador* ou uma *máquina virtual*.

➠ **Questão:** Como codificar as instruções?

- critérios (tipos de instruções, tipos de dados, modelo de execução)
- formatos

Código-máquina e código assembly

➠ O código de um programa pode ser representado por números: *código-máquina*.

Exemplo (em hexadecimal, ARMv7):

```
023081E0
000095E5
046083E4
```

➠ Código simbólico para instruções (mnemónicas): *assembly code*

O mesmo exemplo:

```
add   R3, R2, R1
ldr   R0, [R5]
str   R6, [R3], #4
```

➠ Conversão de código *assembly* para código-máquina também é feita por um programa: *assembler*

➠ O código-máquina difere entre processadores de famílias diferentes. O código-máquina de um Xeon é diferente do código-máquina de um processador Cortex-A5.

Modelo de programação

- O modelo de programação de um microprocessador é definido por:
  1. modelo de execução
  2. conjunto de instruções
     1. classes (ou tipos) de instruções
     2. modos de especificação de operandos (endereçamento)
  3. registos
     1. de uso geral
     2. dedicados (de uso específico)
- Modelo de execução:
  1. inicializar PC *(program counter)*
  2. obter instrução da posição PC da memória
  3. executar instrução e atualizar PC
  4. repetir a partir de 2
- ARMv7: o registo R15 é o PC.

Classes de instruções

➠ As classes de instruções mais comuns são:

1. Operações aritméticas com números inteiros
   - adição, subtração, multiplicação, divisão
2. Operações lógicas sobre conjuntos de bits (números sem sinal)
   - AND, OR, NOR, deslocamentos *(shift)*
3. Transferências de dados
   - leitura e escrita de dados em memória
4. Alteração do fluxo (sequencial) de execução
   - saltos condicionais e comparações
   - saltos incondicionais
   - execução de sub-rotinas

➠ As instruções de salto têm explicitamente a função de alterar o valor do PC.

Modos de endereçamento

➠ Modos de endereçamento = modos de especificação dos operandos

Os mais comuns são:

1. **imediato:** o valor (constante) está incluído na instrução.
2. **registo:** o valor está num registo; a instrução inclui a especificação do registo.
3. **direto:** a instrução inclui o endereço da posição de memória.
4. **indireto** (via registo): o registo contém o endereço da posição de memória onde está o valor; a instrução especifica o registo.
5. **indireto** com deslocamento constante: instrução especifica registo e um valor constante: a posição de memória é obtida por soma do valor constante com o conteúdo do registo.
   (É uma generalização da categoria anterior.)
6. **relativo ao PC**: a instrução inclui constante a adicionar ao valor de PC.

Classificação segundo a origem dos operandos

Mem.	Max. ops.	Arquitetura	Exemplos
0	3	reg-reg	ARM, MIPS, SPARC
1	2	reg-mem	IBM 360/370, Intel 80x86
2	2	mem-mem	VAX
3	3	mem-mem	VAX

Tipo	Vantagens	Desvantagens
reg-reg	Codificação simples, comprimento único. Geração de código simplificada. Duração similar.	Número de instruções elevado. Programas mais compridos.
reg-mem	Acesso a dados sem "load" em separado. Tendem a ter boa densidade de codificação.	Operandos não são equivalentes. Duração varia com a localização dos operandos. Pode restringir o número de registos codificáveis.
mem-mem	Programas compactos. Não ocupa registos com resultados temporários.	Comprimento de instruções muito variável. Complexidade de instruções muito variável. Acesso a memória é crítico.

- As duas principais caraterísticas que diferenciam arquiteturas com registos de uso genérico são:
  1. número de operandos: 2 ou 3
  2. quantos operandos podem residir em memória: de 0 a 3

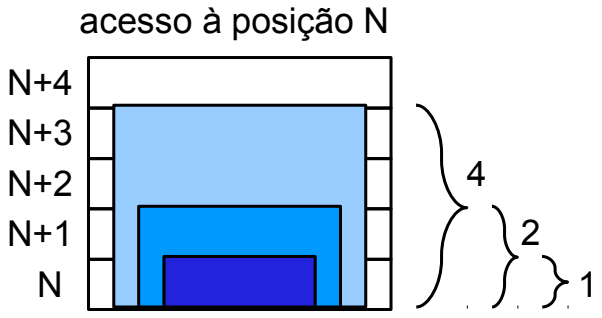
Tipos de operandos

- Tipos comuns de operandos:

1. números inteiros de:
   - 4 bytes (1 palavra)
   - 2 bytes (meia palavra, *half-word*)
   - 1 byte
2. números de vírgula flutuante:
   - 4 bytes (precisão simples *single, float*)
   - 8 bytes (precisão dupla, *double*)

- A interpretação dos dados e o seu tamanho são definidos pela instrução usada para os processar. O programador e/ou o compilador são responsáveis pela utilização coerente das instruções.

- Endereço de memória do item especifica **a posição do primeiro byte**.



- Regras de **alinhamento** típicas:
  - palavra: só endereços múltiplos de 4
  - meia palavra: só endereços múltiplos de 2
  - byte: qualquer endereço

1 Arquitetura do conjunto de instruções

2 Codificação de instruções ARMv7

3 Programação em Assembly

4 Definição e utilização de sub-rotinas

Caraterísticas das instruções ARMv7

- Conjunto de instruções reduzido (RISC = **R**educed **I**nstruction **S**et **C**omputer)
- Organização **reg-reg**
- Acesso a memória:
  - apenas **ldr** (leitura: CPU $\leftarrow$ MEM) e **str** (escrita: MEM$\leftarrow$ CPU)
- Instruções lógicas e aritméticas com 3 registos (2 operandos e 1 resultado)
- Conjunto de instruções "ortogonal":
  - Onde pode ser usado um registo, pode ser usado qualquer outro (quase sempre).
- Todas as instruções têm 32 bits de comprimento
- Endereços válidos: 2^{32} bytes (2^{30} palavras)
- 16 registos (0-15) de 32 bits: `0`, `1`, etc.

Uso especial: 15 $\equiv$ PC, 14 $\equiv$ LR (Link register) para sub-rotinas

- Registo de estado: CPSR *(Current Processor Status Register)*

Utilização convencional dos registos

➠ Sub-rotinas devem ser escritas de forma independente da sua invocação/utilização (por programas **escritos separadamente**): modularidade.

➠ A **interoperabilidade** das sub-rotinas requer o uso de **convenções de utilização dos registos** *(calling conventions)*, que variam com o conjunto de instruções e, possivelmente, com o sistema operativo usado.
Estas regras fazem parte da *interface binária de programas* (ABI = *Application Binary Interface)*

➠ Convenções de utilização de registos numa sub-rotina para ARMv7:

- `0`-`3`: uso sem restrições; são usados para passar os argumentos de uma sub-rotina
  `0` contém o resultado da sub-rotina.
- 4-9: conteúdo deve ser preservado (valor inicial igual a valor final)
- 13: reservado para gestão de uma pilha de dados.
- LR guarda o endereço de retorno de uma sub-rotina

Subconjunto de instruções armV7a (I)

Operação	Sintaxe	Significado
adição	`add dest, op1, op2`	dest ←op1 + op2
subtração	`sub dest, op1, op2`	dest ←op1 - op2
subtração inversa	`rsub dest, op1, op2`	dest ←op2 - op1
E-lógico bit-a-bit	`and dest, op1, op2`	dest ←op1 AND op2
OU-lógico bit-a-bit	`orr dest, op1, op2`	dest ←op1 OR op2
OU exclusivo bit-a-bit	`eor dest, op1, op2`	dest ←op1 XOR op2
E-lógico e negação bit-a-bit	`bic dest, op1, op2`	dest ←op1 AND (NOT op2)
deslocamento lógico para a esquerda	`lsl dest,op1, op2`	dest ←op1 << op2
deslocamento lógico para a direita	`lsr dest,op1, op2`	dest ←op1 >> op2
deslocamento aritmético para a direita	`asr dest,op1, op2`	dest ←op1 >> op2 (sinal)
rotação para a direita	`ror dest,op1, op2`	
rotação para a direita com carry	`rrx dest,op1, op2`	
transferência	`mov dest, op1`	dest ←op1
transferência e negação	`mvn dest, op1`	dest ←NOT(op1)
carregar endereço	`adr dest, etiqueta`	dest ←endereço associado à etiqueta

Subconjunto de instruções ARMv7A (II)

Operação	Sintaxe	Significado
transf. de memória	`ldr dest,[op1,offset]`	dest ←Mem[op1+offset]
tranf. de memória (byte)	`ldrb dest,[op1,offset]`	dest ←Mem[op1+offset]
transf. para memória	`str fonte,[op1,offset`	Mem[op1+offset] ←fonte
transf. para memória (byte)	`strb fonte,[op1,offset`	Mem[op1+offset] ←fonte
comparação aritmética	`cmp op1,op2`	flags como em op1-op2
comparação negada	`cmn op1,op2`	flags como em op1+op2
comparação lógica	`tst op1,op2`	flags como em a op1 AND op2
comparação igualdade lógica	`teq op1,op2`	flags como em op1 XOR op2
salto incondicional	`b alvo`	PC ←alvo
salto condicional	`b{cond} alvo`	se {cond} é verdadeira, PC ←alvo

➠ Os valores de {cond} estão na página seguinte. Os valores dos indicadores (*flags*)) são afetados pelas instruções de comparação.

➠ As instruções começam **sempre** em posições cujos endereços são **múltiplos de 4**.

O registo de indicadores

➠ O registo CPSR contém quatro bits que podem ser afetados pelo resultado de uma instrução. (Indicadores de condição ou *flags*)

Nome	Comportamento
N	N ←1 quando os resultado da operação é negativo, senão N ←0.
Z	Z ←1 quando os resultado da operação é 0, senão N ←0.
C	C ←1 quando a operação resulta em transporte do MSB, senão N ←0.
V	V ←1 se a operação resulta em overflow, senão N ←0.

Os sufixos {cond} correspondem às seguintes condições:

Sufixo	*Flags*	Significado	Sufixo	*Flags*	Significado
EQ	Z= 1	igual	VC	V=0	sem overflow
NE	Z=0	diferente	HI	C=1 e Z=0	maior (sem sinal)
CS ou HS	C=1	maior ou igual (s/s)	LS	C=0 ou Z=1	menor ou igual (s/s)
CC ou LO	C = 0	menor que (s/s)	GE	N=V	maioir ou igual (c/s)
MI	N=1	negativo	LT	N!=V	menor que (c/s)
PL	N=0	positivo ou 0	T	Z=0 e N=V	maior que (c/s)
VS	V=1	overflow (c/s)	LE	Z=1 e N!=V	menor ou igual (c/s)