Sistemas Embebidos

Instruction set

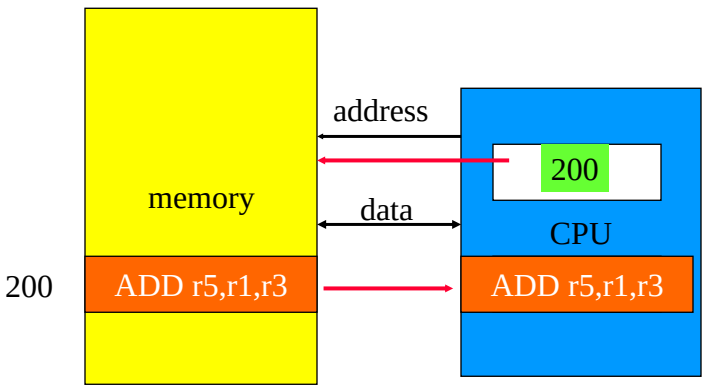
Instruction set - Sumário

- Taxonomia da arquitetura de computadores
- Linguagem *Assembly ARM*

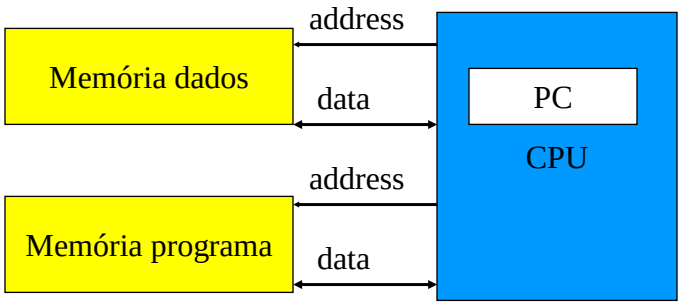
Arquitetura de *von Neumann*

- A memória armazena os dados e instruções
- A unidade de processamento central (*CPU*) *fetches* as instruções ou dados da memória
  - Memória e *CPU* separados distinguem os computadores programáveis
- Suporte dos registos do CPU: *program counter (PC), instruction register (IR), general-purpose registers, etc.*

CPU + memória



Arquitetura *Harvard*



Arquitetura de Harvard contem
2 tipos de memórias separadas.

von Neumann vs. Harvard

- *Harvard* não pode utilizar código adaptativo[1]
- *Harvard* permite dois *fetches* de memória simultâneos
- A maioria dos *DSPs* utiliza a arquitetura *Harvard* para transmissão de dados:
  - maior largura de banda de memória;
  - largura de banda mais previsível.

(1) – Adaptação dinâmica do código em função dos inputs

RISC vs *CISC*

- *Reduced Instruction Set Computer* (*RISC*):
  - *tira maior partido do sotfware*
  - *acesso à memória via load/store*;
  - instruções *pipelinable* e simples
- *Complex Instruction Set Computer* (*CISC*):
  - *maior uso do hardware*
  - elevado número de modos de endereçamento;
  - instruções complexas

Caraterísticas do *instruction set*

- Fixo vs. dimensão variável
- Modos de endereçamento
- Número de operandos
- Tipos de operandos

Modelo de programação

- Os registos disponíveis para utilização por parte dos programas é chamado de programmer/programming model
- Modelo de programação
  - registos visíveis ao programador.
  - Alguns registos não são visíveis *(IR)**.

* Internal Registers

Implementações múltiplas

- As arquiteturas bem sucedidas possuem várias implementações:
  - Diferentes velocidades de relógio;
  - Barramentos de diferentes limites;
  - Diferentes dimensões de cache;
  - etc.

Linguagem *Assembly*

- Características básicas:
  - Uma instrução por linha;
  - *Etiquetas(labels*) - permitem atribuir nomes a endereços (normalmente na 1ª coluna);
  - As instruções começam habitualmente nas colunas seguintes às etiquetas;
  - Comentários são definidos por carateres tipo ";" até ao final da linha.

11

Exemplo da linguagem assembly

```
label1  ADR r4,c
        LDR r0,[r4] ; a comment
        ADR r4,d
        LDR r1,[r4]
        SUB r0,r0,r1 ; comment
```

Etiquetas | Instruções | ; Comentários

12

Pseudo-Operações

- Algumas diretivas do *assembly* não correspondem diretamente a instruções:
  - Definem o endereço atual.
  - Reservam memória.
  - Constantes.
- São instruções para o assembler que não geram nenhum código de máquina.

13

Sistemas embebidos

ARM instruction set

14

ARM instruction set - Sumário

- Versões do ARM
- Linguagem *assembly* do ARM
- Modelo de programação do ARM
- Organização de memória do ARM
- Operações de dados do ARM
- Fluxo de controlo do ARM

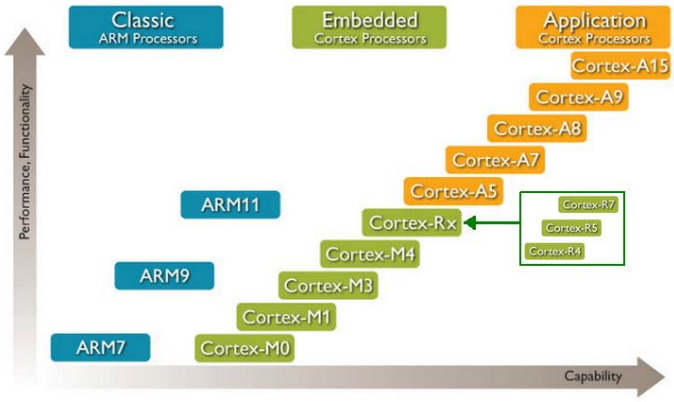
15

Versões do *ARM*

- A arquitetura do *ARM* tem sido estendida sobre várias versões.
- Introduzido no mercado em 1994
- *ARM7* é a família de processadores embebidos de 32-bits mais amplamente utilizada em todo mundo
- Iremos estudar o *ARM7*

16

Migração de design do ARM7



Cortex-Mx – Sistemas embebidos/moveis
Cortex-Ax – aplicações (SO, browsers)
Cortex-Rx – Operações de tempo real

17

Modelo de programação *ARM*

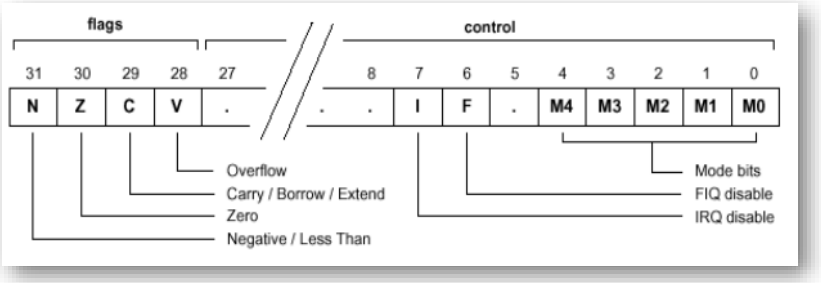
r0	r8	
r1	r9	
r2	r10	
r3	r11	
r4	r12	
r5	r13 (SP)	Stack Pointer
r6	r14 (LR)	Link Register
r7	r15 (PC)	Program counter
	CPSR	Current Program Status Register

- Além dos 16 registos, existe um registo especial, o **CPSR,** este registo é colocado automaticamente durante todas as operações de aritmética, logica e deslocamento…

18

bits/flags de status do *ARM*

CPSR - *Current Program Status Register*



Os principais 4 bits/flags são N Z C V

N - Quando o resultado é negativo;
Z - Quando todos os bits do resultado são "0";
C - Quando existe um transporte de operação;
V - Quando o resultado é um *overflow*.

bits/flags de status do *ARM*

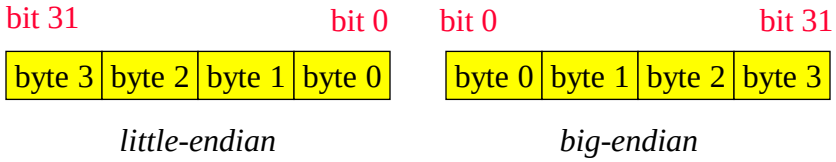
- Todas as operações aritméticas, lógicas, ou de deslocamento definem bits *CPSR*:
  - *N (negativo), Z (zero), C (carry), V (overflow)*

Exemplos:

- -1 + 1 = 0: NZCV = 0110.
- 2^{31}-1+1 = -2^{31}: NZCV = 1001.

Endianness (ordenação)

- Relação entre a ordenação de bit e byte/word define a *endianness*:



Endianess - permite uma maior otimização no processamento, maior performance, dependendo da implementação

Tipo de dados ARM

- *Word* é de 32 bits de comprimento
- *Word* pode ser dividida em quatro bytes de 8-bits
- Os endereços do ARM podem ser de 32 bits de comprimento
- Endereço refere-se ao byte.
- Pode ser configurado no arranque no modo *little-endian* ou *big-endian*

Instruções de dados do *ARM*

- Formato básico:

  `ADD r0,r1,r2`

  - Computa r1+r2, e armazena o resultado em r0.
- Operando imediato:

  `ADD r0,r1,#2`

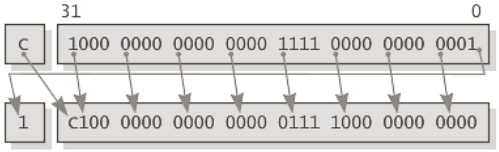
  - Computa r1+2, e armazena o resultado em r0.

Instruções de dados do ARM

- ADD, ADC: add (w. carry)
- SUB, SBC: subtract (w. carry)
- RSB, RSC: reverse subtract (w. carry)
- MUL, MLA: multiply (and accumulate)
- AND, ORR, EOR: Logical AND, OR, Exclusive OR
- BIC: bit clear
- LSL, LSR: logical shift left/right
- ASL, ASR: arithmetic shift left/right
- ROR: rotate right
- RRX: rotate right extended with C

Variações de operações de dados

- *Logical shift*:
  - preenche com "0"s.
- *Arithmetic shift*:
  - preenche com "1"s.
- *RRX* executa uma rotação de 33-bit, incluindo o bit C do *CPSR* acima do bit de sinal.



Instruções de comparação *ARM*

- CMP : compare
- CMN : negated compare
- TST : bit-wise test
- TEQ : bit-wise negated test

Estas instruções definem apenas os bits NZCV do CPSR

Instruções de "move" do *ARM*

- MOV, MVN : move (negated)

```
MOV r0, r1 ; sets r0 to r1
```


27

Instruções de *load/store* do *ARM*

- *LDR, LDRH, LDRB : load (half-word, byte)*
- *STR, STRH, STRB : store (half-word, byte)*
- Modos de endereçamento:
  - Registo indireto : `LDR r0,[r1]`
  - Com segundo registo : `LDR r0,[r1,-r2]`
  - Com constante : `LDR r0,[r1,#4]`

28

Exemplo: atribuições em C

- C:

```
x = (a + b) - c;
```

- Assembler:

```
ADR r4,a        ; get address for a (atribui o endereço a "a")
LDR r0,[r4]     ; get value of a (carrega o valor de "a")
ADR r4,b        ; get address for b, reusing r4
LDR r1,[r4]     ; get value of b
ADD r3,r0,r1    ; compute a+b
ADR r4,c        ; get address for c
LDR r2,[r4]     ; get value of c
SUB r3,r3,r2    ; complete computation of x
ADR r4,x        ; get address for x
STR r3,[r4]     ; store value of x
```


Exemplo: atribuições C

- C:

```
y = a*(b+c);
```

- Assembler:

```
ADR r4,b ; get address for b
LDR r0,[r4] ; get value of b
ADR r4,c ; get address for c
LDR r1,[r4] ; get value of c
ADD r2,r0,r1 ; compute partial result
ADR r4,a ; get address for a
LDR r0,[r4] ; get value of a
MUL r2,r2,r0 ; compute final value for y
ADR r4,y ; get address for y
STR r2,[r4] ; store y
```

Exemplo: atribuições C

- C:

```
z = (a << 2) |  (b & 15);
```

- Assembler:

```
ADR r4,a ; get address for a
LDR r0,[r4] ; get value of a
MOV r0,r0,LSL 2 ; perform shift (realiza deslocamento)
ADR r4,b ; get address for b
LDR r1,[r4] ; get value of b
AND r1,r1,#15 ; perform AND (realiza operação lógica AND)
ORR r1,r0,r1 ; perform OR(realiza operação lógica OR)
ADR r4,z ; get address for z
STR r1,[r4] ; store value for z
```


31

Fluxo de controlo do ARM

- Todas as operações podem ser executadas condicionalmente, testando o *CPSR*:
  - EQ, NE, CS, CC, MI, PL, VS, VC, HI, LS, GE, LT, GT, LE
- *Branch* (salto):

```
B #100 ; salto para a posição 100
```

  - Pode ser executada condicionalmente

32

fluxo de controlo do ARM

EQ	Equals zero	Z = 1
NE	Not equal to zero	Z = 0
CS	Carry set	C = 1
CC	Carry clear	C = 0
MI	Minus	N = 1
PL	Nonnegative (plus)	N = 0
VS	Overflow	V = 1
VC	No overflow	V = 0
HI	Unsigned higher	C = 1 and Z = 0
LS	Unsigned lower or same	C = 0 or Z = 1
GE	Signed greater than or equal	N = V
LT	Signed less than	N V
GT	Signed greater than	Z = 0 and N = V
LE	Signed less than or equal	Z = 1 or N V

Exemplo: estrutura if

- C:

```
if (a < b){
        x = 5;
        y = c + d;
        }
else x = c - d;
```

- Assembler:

```
; compute and test condition
  ADR r4,a ; get address for a
  LDR r0,[r4] ; get value of a
  ADR r4,b ; get address for b
  LDR r1,[r4] ; get value for b
  CMP r0,r1 ; compare a < b
  BGE fblock ; if a >= b, branch to false block
```

estrutura if, cont.

```
if (a < b){
      x = 5;
      y = c + d;
      }
else x = c - d;
```

```
; true block
  MOV r0,#5 ; generate value for x
  ADR r4,x ; get address for x
  STR r0,[r4] ; store x
  ADR r4,c ; get address for c
  LDR r0,[r4] ; get value of c
  ADR r4,d ; get address for d
  LDR r1,[r4] ; get value of d
  ADD r0,r0,r1 ; compute y
  ADR r4,y ; get address for y
  STR r0,[r4] ; store y
  B after ; branch around false block
```


estrutura if, cont.


```
if (a < b){
      x = 5;
      y = c + d;
      }
else x = c - d;
```


```
; false block
fblock ADR r4,c ; get address for c
       LDR r0,[r4] ; get value of c
       ADR r4,d ; get address for d
       LDR r1,[r4] ; get value for d
       SUB r0,r0,r1 ; compute c-d
       ADR r4,x ; get address for x
       STR r0,[r4] ; store value of x
after ...
```

Sumário

- Arquitetura de *load/store*
- A maioria das instruções são RISCy, operam num ciclo apenas
  - Algumas operações "multi-register" demoram mais tempo
- Todas as instruções podem ser executadas condicionalmente

37

Exercícios

- Considere as seguintes atribuições C e escreva-as para Assembler do ARM:
  - x= a * b - c;
  - y= (a|15) & (3 >> b);
  - if (a>=b) x=7; else x=c-d;

38