HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

BÀI GIẢNG MÔN

KỸ THUẬT VI XỬ LÝ

Giảng viên: TS. Vũ Hữu Tiến
Điện thoại/E-mail: 0932357079 / tienvh@ptit.edu.vn
Học kỳ/Năm biên soạn: Kỳ 1/2014

KỸ THUẬT VI XỬ LÝ

NỘI DUNG

- Chương 1 – Tổng quan về hệ vi xử lý
- Chương 2 – Bộ vi xử lý ARM
- Chương 3 – Lập trình hợp ngữ cho vi xử lý ARM
- Chương 4 – Vi điều khiển 8051
- Chương 5 – Bộ đếm/định thời và UART trong 8051
- Chương 6 – Lập trình ngắt trong 8051

2

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

NỘI DUNG

1. Giới thiệu về lập trình Assembly
2. Tập lệnh của ARM
   1. Lệnh xử lý dữ liệu (data processing)
   2. Dịch chuyển dữ liệu (data movement)
   3. Điều khiển chương trình (flow control)
   4. Ngắt

3

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Cấu trúc chung của một chương trình hợp ngữ:

```
        AREA Example, CODE, READONLY      ; name this block of code
        ENTRY                             ; mark first instruction
                                          ; to execute
start
        MOV     r0, #15                   ; Set up parameters
        MOV     r1, #20
        BL      firstfunc                 ; Call subroutine
        SWI     0x11                      ; terminate
firstfunc                                 ; Subroutine firstfunc
        ADD     r0, r0, r1                ; r0 = r0 + r1
        MOV     pc, lr                    ; Return from subroutine
                                          ; with result in r0
        END                               ; mark end of file
```

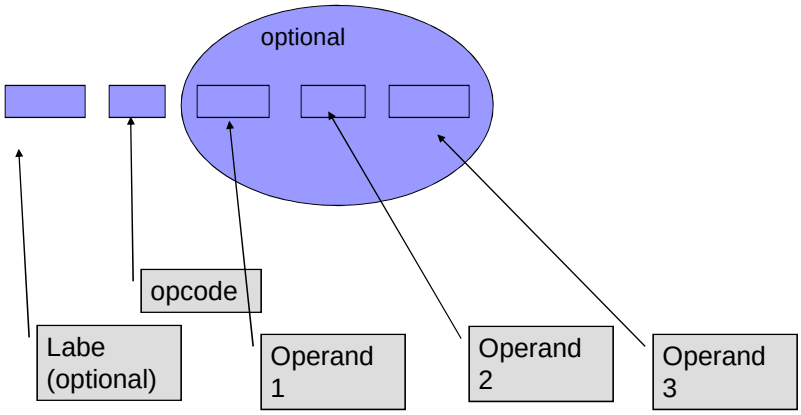
label opcode operands comment

4

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Cấu trúc chung của một lệnh ARM

  label <dấu cách> lệnh <dấu cách> ; chú giải



5

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

NỘI DUNG

1. Giới thiệu về lập trình Assembly
2. Tập lệnh của ARM
   1. Lệnh xử lý dữ liệu (data processing)
   2. Di chuyển dữ liệu (data transfer)
   3. Điều khiển chương trình (flow control)

6

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Đặc điểm của tập lệnh ARM:
  - Kiến trúc Load – Store
  - Lệnh 3 địa chỉ
  - Tất cả đều là cách lệnh có điều kiện
  - Có khả năng load/store nhiều thanh ghi đồng thời
  - Khối dịch và khối ALU có thể hoạt động song song, do đó phép tính dịch và các phép tính tính toán có thể được thực hiện đồng thời.

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh xử lý dữ liệu:
  - Lệnh xử lý dữ liệu bao gồm lệnh di chuyển dữ liệu giữa các thanh ghi (move), lệnh số học (arithmetic), lệnh logic (logical), lệnh so sánh (comparison) và lệnh nhân (multiply).
  - Hầu hết các lệnh xử lý dữ liệu có thể dùng bộ dịch (barrel shifter) để xử lý một trong những toán hạng của lệnh

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh xử lý dữ liệu
  - Arithmetic: **ADD ADC SUB SBC RSB RSC**
  - Logical: **AND ORR EOR BIC**
  - Comparisons: **CMP CMN TST TEQ**
  - Data movement: **MOV MVN**
- Lưu ý: các lệnh chỉ thực hiện trên thanh ghi, KHÔNG thực hiện trên bộ nhớ.
- Cú pháp:

  <Operation>{<cond>}{S} Rd, Rn, Operand2
  - Lệnh so sánh tác động đến cờ và thanh ghi Rd không bị tác động
  - Lệnh di chuyển dữ liệu không tác động đến thanh ghi Rn
- Toán hạng thứ 2 được đưa đến ALU thông qua bộ dịch chuyển

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Di chuyển dữ liệu giữa các thanh ghi

  MOV<cond><S> Rd, Rn, <operands>

```
MOVCS R0, R1   ; Nếu cờ nhớ C = 1 thì R0 := R1

MOVS R0, #0    ; R0 = 0
               ; Z = 1, N = 0
               ; C, V không bị tác động
```

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Di chuyển dữ liệu giữa các thanh ghi:
  - Lưu ý: Hầu hết các lệnh trong ARM có trường điều kiện. Khi thỏa mãn điều kiện đó thì lệnh mới được thực hiện.

MOVCS R0, R1 *; R0 = R1 chỉ khi C = 1.*

MOVCC R0, R1 *; R0 = R1 chỉ khi C = 0.*

Mnemonic	Condition	Mnemonic	Condition
CS	*C*arry *S*et	CC	*C*arry *C*lear
EQ	*E*qual (Zero Set)	NE	*N*ot *E*qual (Zero Clear)
VS	Overflow *S*et	VC	Overflow *C*lear
GT	*G*reater *T*han	LT	*L*ess *T*han
GE	*G*reater Than or *E*qual	LE	*L*ess Than or *E*qual
PL	*P*lus (Positive)	MI	*M*inus (Negative)
HI	*H*igher Than	LO	*L*ower Than (aka CC)
HS	*H*igher or *S*ame (aka CS)	LS	*L*ower or *S*ame

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Di chuyển dữ liệu giữa các thanh ghi:
  - Gồm có hai lệnh:
    - *MOV R0, R2* *; R0 = R2*
    - *MVN R0, R2* *; move negative, R0 = ~ R2*
  - *Ví dụ:*

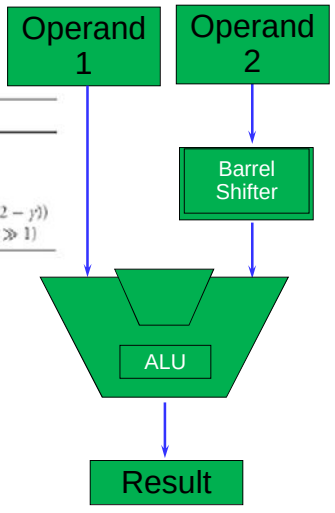
Trước: *r5 = 5*
r7 = 8
MOV r7, r5
Sau: *r5 = 5*
r7 = 5

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Các chế độ địa chỉ:
  - Chế độ địa chỉ thanh ghi:
    - *ADD R0, R1, R2 ; Các toán hạng là các thanh ghi*
  - Chế độ địa chỉ tức thời:
    - ADD R3, R3, #1 ; R3 = R3 + 1
    - ADD R8, R7, #0xff ; R8 = R7[7:0]
- Dấu # được sử dụng để biểu diễn toán hạng tức thời
- Toán hạng có ký hiệu *0x* đứng trước: biểu diễn số hexa

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Barrel shiffter:



Mnemonic	Description	Shift	Result
LSL	logical shift left	*x*LSL *y*	$x \ll y$
LSR	logical shift right	*x*LSR *y*	$(\text{unsigned})x \gg y$
ASR	arithmetic right shift	*x*ASR *y*	$(\text{signed})x \gg y$
ROR	rotate right	*x*ROR *y*	$((\text{unsigned})x \gg y) \mid (x \ll (32 - y))$
RRX	rotate right extended	*x*RRX	$(c \text{ flag} \ll 31) \mid ((\text{unsigned})x \gg 1)$

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Dịch trái:

- MOV R0, R2, LSL #2 ; R0 = R2<<2
  ; R2 không đổi

Ví dụ: 0…0 0011 0000
Trước: R2 = 0x00000030
Sau: R0 = 0x000000C0
R2 = 0x00000030

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Dịch phải:

- MOV R0, R2, LSR #2 ; R0 = R2<<2
  ; R2 không đổi

Ví dụ: 0…0 0011 0000
Trước: R2 = 0x00000030
Sau: R0 = 0x0000000C
R2 = 0x00000030

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Dịch phải số học:

- MOV R0, R2, ASR #2 ; R0 = R2>>2
  ; R2 không đổi

Ví dụ: 1010 0…0 0011 0000
Trước: R2 = 0xA0000030
Sau: R0 = 0xE800000C
R2 = 0xA0000030

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

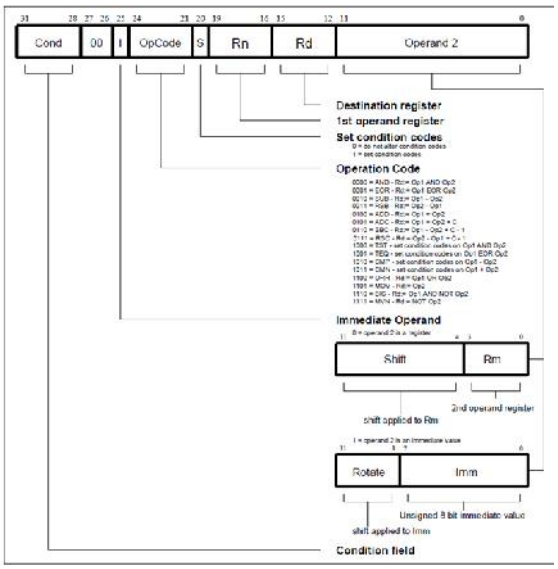
- Quay phải mở rộng:

- MOV R0, R2, RRX ; R0 = R2 sau khi quay
  ; R2 không đổi

Ví dụ: 0…0 0011 0001
Trước: R2 = 0x00000031 ; C = 1
Sau: R0 = 0x80000018 ; C = 1
R2 = 0x00000031

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Cấu trúc lệnh được mã hóa:



CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Các lệnh số học: Cộng và trừ

  instruction<cond><S> Rd, Rn, N

ADC	add two 32-bit values and carry	$Rd = Rn + N +$ carry
ADD	add two 32-bit values	$Rd = Rn + N$
RSB	reverse subtract of two 32-bit values	$Rd = N - Rn$
RSC	reverse subtract with carry of two 32-bit values	$Rd = N - Rn -$!(carry flag)
SBC	subtract with carry of two 32-bit values	$Rd = Rn - N -$!(carry flag)
SUB	subtract two 32-bit values	$Rd = Rn - N$

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Các lệnh số học: Cộng và trừ

Ví dụ 1:

R0 = 0x00000000
R1 = 0x00000002
R2 = 0x00000001
SUB R0, R1, R2

Ví dụ 2:

R0 = 0x00000000
R1 = 0x00000077
RSB R0, R1, #0

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Các lệnh số học: Cộng và trừ

Ví dụ 3:

R1 = 0x00000001
SUBS R1, R1, #1
CPRS = ?

Ví dụ 4:

R0 = 0x00000000
R1 = 0x00000005
ADD R0, R1, R1, LSL #1

Exercise 2

Fill in the shaded areas.

Program counter PC =R15, #value = intermediate constant value

Current Program Status Register (CPSR)

Address (H)		Comments	After instruction is run				
PC			PC (Hex)	C	R0(Hex)	R1(Hex)	R2 (Hex)
	All registers R0-R2 are rest to 0 here			0	0	0	
0000 1000	Mov r1,#15	;r1=15	0000 1004	0	0000 0000	0000 000f	ffff ffff
	Mov r2,#0xffffffff	;r2=#0xffffffff ;i.e. r2= -1					
	ADDs r0,r1,r2	;r0=r1+r2					
	ADCs r0,r1,r2	;r0=r1+r2+C					
	SUBs r0,r1,r2	;r0=r1-r2					
	SBCs r0,r1,r2	;r0=r1-r2+C-1					

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh logic

instruction<cond><S> Rd, Rn, N

AND	logical bitwise AND of two 32-bit values	$Rd = Rn \& N$
ORR	logical bitwise OR of two 32-bit values	$Rd = Rn \mid N$
EOR	logical exclusive OR of two 32-bit values	$Rd = Rn \wedge N$
BIC	logical bit clear (AND NOT)	$Rd = Rn \& \sim N$

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh logic
  - AND R0, R1, R2 ; R0 = R1 and R2
  - ORR R0, R1, R2 ; R0 = R1 or R2
  - EOR R0, R1, R2 ; R0 = R1 xor R2
  - BIC R0, R1, R2 ; R0 = R1 and (~R2)

Lệnh BIC là lệnh xóa bit: Các bit trong R1 sẽ bị xóa bởi các bit được đánh dấu trong R2

R1 = 0x11111111 R2 = 0x01100101

BIC R0, R1, R2

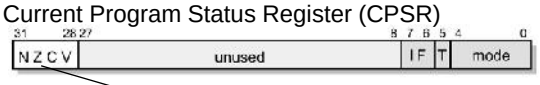
R0 = 0x10011010

Exercise 3

Fill in the shaded areas.

Program counter PC =R15, #value = intermediate constant value

Current Program Status Register (CPSR)



Address (H)		Comments	After instruction is run			
PC			R0(Hex)	R1(Hex)	R2(Hex)	NZ
	At the beginning		0000 0000H	0000 0055H	0000 0061H	00
0000 7000	ANDs r0,r1,r2	;r0=r1 and r2 (bit by bit)				
	ORRs r0,r1,r2	;r0=r1 or r2				
	EORs r0,r1,r2	;r0=r1 xor r2				
	BICs r0,r1,r2	;r0=r1 and (not r2)				

R1=55H=0101 0101 B
R2=61H=0110 0001 B
9EH=1001 1110 B

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh so sánh
  - Các lệnh so sánh không tạo ra kết quả nhưng nó tác động đến các bit cờ (N, Z, C, V) trong thanh ghi CPSR.
  - Cú pháp: instruction<cond> Rn, N

CMN	compare negated	flags set as a result of $Rn + N$
CMP	compare	flags set as a result of $Rn - N$
TEQ	test for equality of two 32-bit values	flags set as a result of $Rn \wedge N$
TST	test bits of a 32-bit value	flags set as a result of $Rn \& N$

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh so sánh
  - CMP R1, R2 ; Thiết lập cờ dựa trên kết quả R1 – R2
  - CMN R1, R2 ; Thiết lập cờ dựa trên kết quả R1 + R2
  - TST R1, R2 ; bit test: Thiết lập cờ dựa trên kq R1 and R2
  - TEQ R1, R2 ; test equal: Thiết lập cờ dựa trên kq R1 xor R2

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh so sánh

CMP r1, r2 ; set cc on r1 - r2 (compare)

- Same as SUB (subtract) except result of subtraction is not stored.
- Only the condition code bits (cc) {N,Z,C,V} in CPSR are changed

•N = 1 if MSB of (r1 - r2) is '1' (MSB of result is sign bit, 1 = negative)
•Z=1 when the result is zero
•C=1 when the result of an addition is greater than or equal to 2^{32}, if the result of a subtraction is positive.
•V=1 (when result of add, subtract, or compare is >= 2^{31}, or < -2^{31}.). I.e.
  •if two -ve numbers are added, the result is +ve (underflow).
  •if two +ve numbers are added, the result is -ve (overflow).
  (0x7FFFFFFF+1=0x80000000)

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh so sánh

Ví dụ:

```
R1 = a, R2 = b
if (R1<R2)
        R1=a
else
        R1=b
```

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

Exercise 6

Fill in the shaded areas.

TST updates the N and Z flags according to the result, It does not affect the C or V flags.

Address (H)		Comments	After instruction is run		
PC			NZCV (binary)	R1 (Hex)	R2 (Hex)
	All registers R0-R2=0 and NZCV=0000, here				
0000 1000	Mov r1,#15	;r1=15 decimal			
	Mov r2,#0x240	;r2=0xF0 (0xf is 240 in decimal)			
	TST r1,r2	; set cc on r1 AND r2 (logical AND operation test bits)			
	TEQ r1,r2	; set cc on r1 xor r2 (test equivalent)			

Convert hex to decimal :http://easycalculation.com/hex-converter.php

0000 1111 0000 1111

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh nhân

  MLA<cond><S> Rd, Rm, Rs, Rn

  MUL<cond><S> Rd, Rm, Rs

MLA	multiply and accumulate	$Rd = (Rm^*Rs) + Rn$
MUL	multiply	$Rd = Rm^*Rs$

- Tất cả các toán hạng phải là thanh ghi
- Thanh ghi Rm, Rs phải là hai thanh ghi khác nhau

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh nhân 64 bit

  instruction<cond><S> RdLo, RdHi, Rm, Rs

SMLAL	signed multiply accumulate long	[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)
SMULL	signed multiply long	[RdHi, RdLo] = Rm * Rs
UMLAL	unsigned multiply accumulate long	[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)
UMULL	unsigned multiply long	[RdHi, RdLo] = Rm * Rs

r0 = 0x00000000 r1 = 0x00000000

r2 = 0xf0000002 r3 = 0x00000002

EMULL r0,r1,r2,r3

r0 = 0xe0000004 r1 = 0x00000001

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh nhân 64 bit

  instruction<cond><S> RdLo, RdHi, Rm, Rs

SMLAL	signed multiply accumulate long	[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)
SMULL	signed multiply long	[RdHi, RdLo] = Rm * Rs
UMLAL	unsigned multiply accumulate long	[RdHi, RdLo] = [RdHi, RdLo] + (Rm * Rs)
UMULL	unsigned multiply long	[RdHi, RdLo] = Rm * Rs

r0 = 0x00000000 r1 = 0x00000000

r2 = 0xf0000002 r3 = 0x00000002

EMULL r0,r1,r2,r3

r0 = 0xe0000004 r1 = 0x00000001

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

NỘI DUNG

1. Giới thiệu về lập trình Assembly
2. Tập lệnh của ARM
   1. Lệnh xử lý dữ liệu (data processing)
   2. Điều khiển chương trình (flow control)
   3. Di chuyển dữ liệu (data transfer)
   4. Ngắt

35

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh rẽ nhánh
  - Lệnh rẽ nhánh không điều kiện

```
        B       label
        ….
label:  ….
```

  - Lệnh rẽ nhánh có điều kiện

```
        MOV  R0, #0
loop:   ADD  R0, R0, #1
        CMP  R0, #10
        BNE  loop
```

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh rẽ nhánh
  - Lệnh rẽ nhánh không điều kiện

```
        B       label
        ….
label:  ….
```

  - Lệnh rẽ nhánh có điều kiện

```
        MOV  R0, #0
loop:   ADD  R0, R0, #1
        CMP  R0, #10
        BNE  loop
```

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh rẽ nhánh

Branch	Interpretation	Normal uses
B BAL	Unconditional Always	Always take this branch Always take this branch
BEQ	Equal	Comparison equal or zero result
BNE	Not equal	Comparison not equal or non-zero result
BPL	Plus	Result positive or zero
BMI	Minus	Result minus or negative
BCC BLO	Carry clear Lower	Arithmetic operation did not give carry-out Unsigned comparison gave lower
BCS BHS	Carry set Higher or same	Arithmetic operation gave carry-out Unsigned comparison gave higher or same
BVC	Overflow clear	Signed integer operation: no overflow occurred
BVS	Overflow set	Signed integer operation: overflow occurred
BGT	Greater than	Signed integer comparison gave greater than
BGE	Greater or equal	Signed integer comparison gave greater or equal
BLT	Less than	Signed integer comparison gave less than
BLE	Less or equal	Signed integer comparison gave less than or equal
BHI	Higher	Unsigned comparison gave higher
BLS	Lower or same	Unsigned comparison gave lower or same

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh rẽ nhánh

```
          CMP   R0, #5
          BEQ   bypass      @ if (R0!=5) {
          ADD   R1, R1, R0  @  R1=R1+R0-R2
          SUB   R1, R1, R2  @ }
bypass:   …
-------------------------------
          CMP   R0, #5        smaller and faster
          ADDNE R1, R1, R0
          SUBNE R1, R1, R2
```

Rule of thumb: if the conditional sequence is three instructions or less, it is better to use conditional execution than a branch.

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh rẽ nhánh

```
    if ((R0==R1) && (R2==R3)) R4++
-------------------------------
          CMP   R0, R1
          BNE   skip
          CMP   R2, R3
          BNE   skip
          ADD   R4, R4, #1
skip:     …
-------------------------------
          CMP   R0, R1
          CMPEQ R2, R3
          ADDEQ R4, R4, #1
```

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh gọi hàm
  - Lệnh BL copy địa chỉ quay trở về chương trình chính vào thanh ghi R14 (lr)

```
        BL     subfunc      ; gọi hàm subfunc
        CMP    R1, #5       ; vị trí quay trở lại
        …..
Subfunc: …..                ; hàm con
        …..
        MOV PC, LR          ; quay lại chương trình chính
```

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

NỘI DUNG

1. Giới thiệu về lập trình Assembly
2. Tập lệnh của ARM
   1. Lệnh xử lý dữ liệu (data processing)
   2. Điều khiển chương trình (flow control)
   3. Di chuyển dữ liệu (data transfer)
   4. Ngắt

42

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Lệnh di chuyển dữ liệu
  - Di chuyển dữ liệu giữa các thanh ghi và bộ nhớ
  - Có 3 dạng chính:
    - Load/store một thanh ghi
    - Load/store nhiều thanh ghi
    - Hoán chuyển dữ liệu giữa ô nhớ và thanh ghi

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store một thanh ghi
  - Cú pháp:
    - <LDR|STR>{cond}{B}, address
    - LDR{cond}SB|H|SH Rd, address
    - STR{cond}H|Rd, address
  - Độ dài dữ liệu có thể là 1 byte, một từ 32 bit, một nửa từ 16 bit.

```
LDR   r0, [r1]    ; r0 := mem32[r1]
STR   r0, [r1]    ; mem32[r1] := r0
```

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

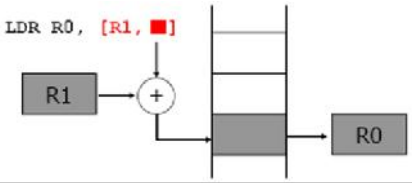
- Load/Store một thanh ghi

LDR	Load a word into register	Rd ←mem32[address]
STR	Store a word in register into memory	Mem32[address] ←Rd
LDRB	Load a byte into register	Rd ←mem8[address]
STRB	Store a byte in register into memory	Mem8[address] ←Rd
LDRH	Load a half-word into register	Rd ←mem16[address]
STRH	Store a half-word in register into memory	Mem16[address] ←Rd
LDRSB	Load a signed byte into register	Rd ←signExtend(mem8[address])
LDRSH	Load a signed half-word into register	Rd ←signExtend(mem16[address])

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store một thanh ghi
  - Chế độ địa chỉ Pre-indexed
    - Địa chỉ bao gồm địa chỉ cơ sở (lưu trong thanh ghi) và các địa chỉ đoạn (offset)
    - Mục đích: cho phép truy cập tới các vị ô nhớ trong một vùng nhớ.

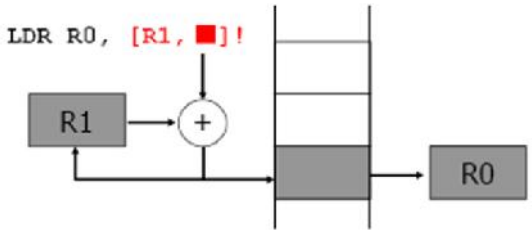
```
LDR     R0, [R1, #4]      ; R0 = mem[R1+4]
                          ; R1 không đổi
```



CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store một thanh ghi
  - Chế độ địa chỉ Auto-indexing (Preindex with writeback)

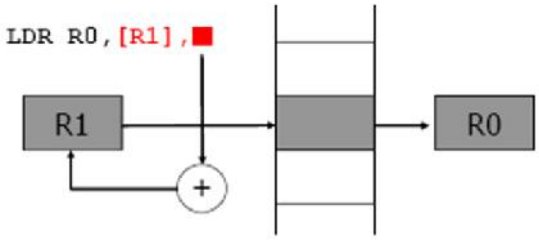
```
LDR R0, [R1,#4]        ; R0 = mem[R1+4]
                       ; R1 = R1 + 4
```



CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store một thanh ghi
  - Chế độ địa chỉ Post-indexed
    - Không sử dụng dấu chấm than (!)

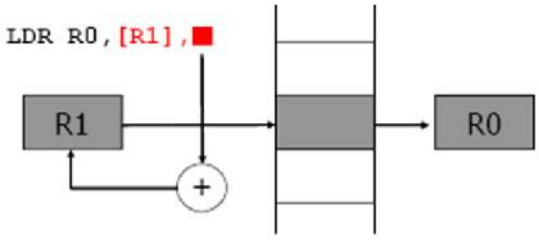
```
LDR    R0, R1, #4      ; R0 = mem[R1]
                       ; R1 = R1 + 4
```



CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store một thanh ghi
  - Chế độ địa chỉ Post-indexed
    - Không sử dụng dấu chấm than (!)

```
LDR     R0, R1, #4      ; R0 = mem[R1]
                        ; R1 = R1 + 4
```



CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store nhiều thanh ghi
  - Cho phép một lượng lớn dữ liệu được trao đổi đồng thời

```
LDMIA   r1, {r0, r2, r5}    ; r0 := mem32[r1]
                            ; r2 := mem32[r1 + 4]
                            ; r5 := mem32[r1 + 8]
```

The base register r1 should be word-aligned

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store nhiều thanh ghi
  - Cho phép một lượng lớn dữ liệu được trao đổi đồng thời

LDM	Load multiple registers
STM	Store multiple registers

Addressing mode	Description	Starting address	End address	Rn!
IA	Increment After	Rn	Rn+4*N-4	Rn+4*N
IB	Increment Before	Rn+4	Rn+4*N	Rn+4*N
DA	Decrement After	Rn-4*Rn+4	Rn	Rn-4*N
DB	Decrement Before	Rn-4*N	Rn-4	Rn-4*N

Addressing mode for multiple register load and store instructions

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Load/Store nhiều thanh ghi
  - Cho phép một lượng lớn dữ liệu được trao đổi đồng thời

LDM	Load multiple registers
STM	Store multiple registers

Addressing mode	Description	Starting address	End address	Rn!
IA	Increment After	Rn	Rn+4*N-4	Rn+4*N
IB	Increment Before	Rn+4	Rn+4*N	Rn+4*N
DA	Decrement After	Rn-4*Rn+4	Rn	Rn-4*N
DB	Decrement Before	Rn-4*N	Rn-4	Rn-4*N

Addressing mode for multiple register load and store instructions

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Hoán chuyển dữ liệu giữa ô nhớ và thanh ghi

```
SWP{B}   Rd, Rm, [Rn]
```

SWP	WORD exchange	tmp = mem32[Rn] mem32[Rn] = Rm Rd = tmp
SWPB	Byte exchange	tmp = mem8[Rn] mem8[Rn] = Rm Rd = tmp

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

NỘI DUNG

1. Giới thiệu về lập trình Assembly
2. Tập lệnh của ARM
   1. Lệnh xử lý dữ liệu (data processing)
   2. Điều khiển chương trình (flow control)
   3. Di chuyển dữ liệu (data transfer)
   4. Ngắt

54

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Ngắt mềm:
  - Ngắt mềm (software interrupt) tạo ra một biệt lệ cho phép ứng dụng gọi một số tác vụ của hệ điều hành
  - Cú pháp: SWI{cond} SWI_number

SWI	software interrupt	*lr_svc* = address of instruction following the SWI *spsr_svc* = *cpsr* *pc* = vectors + 0x8 *cpsr* mode = *SVC* *cpsr I* = 1 (mask IRQ interrupts)

CHƯƠNG 3-LẬP TRÌNH HỢP NGỮ ARM

- Ngắt mềm: