Chapter 2

Data Organisation in Memory

Quiz Questions in Video Lectures

CZ1106
CE1106

Chapter 2

Data Organisation in Memory

Role of Memory in Computing

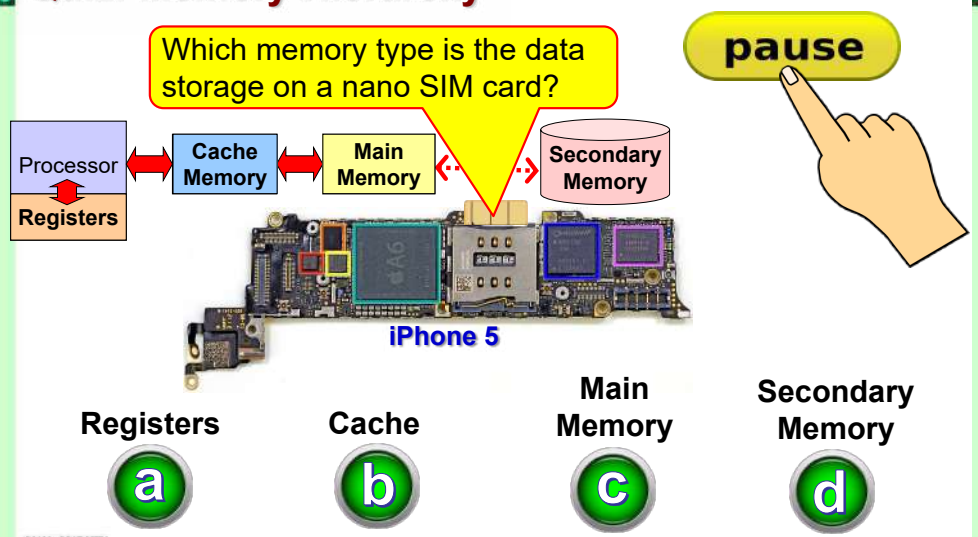
Learning Objectives (2.1)

1. Describe the concept of programming in software.
2. Describe the von Neumann's stored program concept.
3. Describe the role of memory in computing.
4. Describe the characteristics and function of different data storage elements in the memory hierarchy.

©2020, SCSE/NTU

CZ1106
CE1106

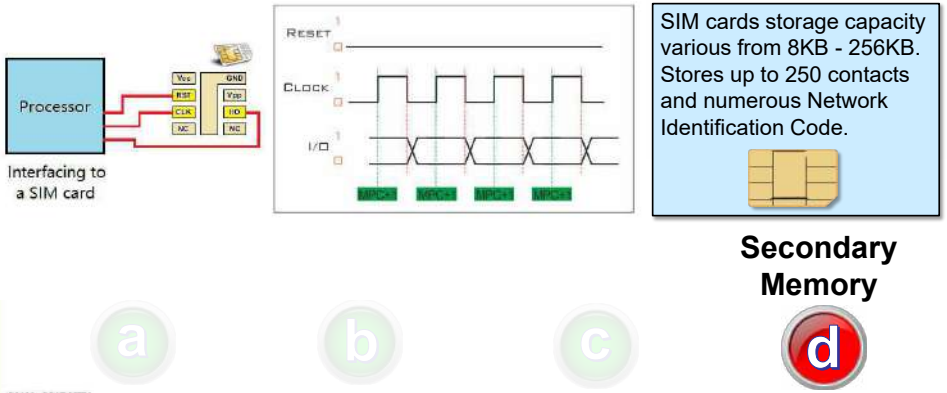
Quiz: Memory Hierarchy



©2020, SCSE/NTU

CZ1106
CE1106

Quiz: Memory Hierarchy



©2020, SCSE/NTU

Chapter 2

Data Organisation in Memory

Number Representation

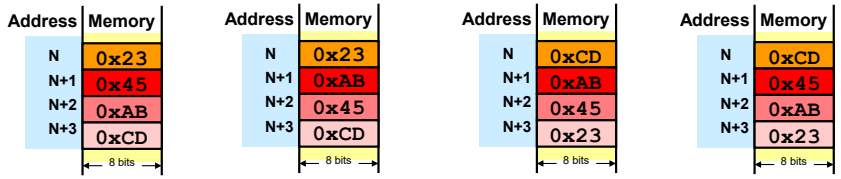
Learning Objectives (2.2)

1. Describe the different C numeric data types and their characteristics.
2. Describe the concept of numeric range and its implications to data size.
3. Describe how multi-byte numbers are stored in memory.

Quiz: Byte-ordering

- How would a 32-bit hexadecimal value of `0x23AB45CD` be stored in memory using the Big Endian format?

pause



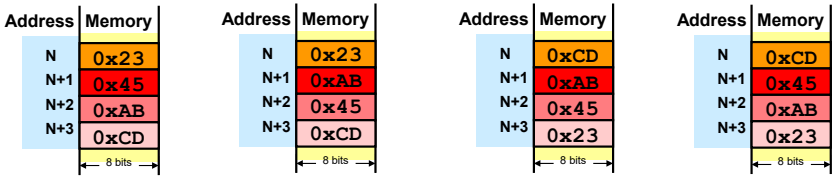
a b c d

Quiz: Byte-ordering

MSB at the top

- How would a 32-bit hexadecimal value of `0x23AB45CD` be stored in memory using the Big Endian format?

In Big Endian, the **most significant** byte (i.e. `0x23`) is stored at the **lowest** address. The rest the bytes are ordered into consecutive increasing memory addresses.



a b c d

Quiz: Defining Variable Size

- How many bytes are allocated for an `unsigned long long int` variable?

pause

4 8 32 64

a b c d

CZ1106 CE1106

Quiz: Defining Variable Size

- How many bytes are allocated for an **`unsigned long long int`** variable?

same for signed long long int

The **`unsigned long long int`** is an unsigned integer represented by 8 bytes. This is equivalent to 64 bits of data. If you have selected 64, you did not look at the question carefully.

64 **32** **8** **4**

a b c d

CZ1106 CE1106

Chapter 2

Data Organisation in Memory

Character and Boolean Representation

Learning Objectives (2.3)

1. Describe the char data type and its representations.
2. Describe the Boolean data type and its implementation.

CZ1106 CE1106

Quiz: Text File in Memory

- Which of the memory contents below is **least likely** to be a segment of memory containing a plain text file?

pause

MS / LS	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	
F	SI	US	/	?	O	_	o	DEL

7-bit ASCII Table

Address	Memory	
N	0x4E	N
N+1	0x6F	O
N+2	0x0D	CR
N+3	0x0A	LF

a

Address	Memory
N	0x2B
N+1	0x36
N+2	0x35
N+3	0x20

b

Address	Memory	
N	0x42	
N+1	0x45	
N+2	0x4C	
N+3	0x07	BEL

c

control character; cannot be printed (suppose to be not)

CZ1106 CE1106

Quiz: Text File in Memory

- Which of the memory contents below is **least likely** to be a segment of memory containing a plain text file?

MS / LS	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	
F	SI	US	/	?	O	_	o	DEL

7-bit ASCII Table

No CR LF

Address	Memory	
N	0x4E	"N"
N+1	0x6F	"o"
N+2	0x0D	CR
N+3	0x0A	LF

a

+65 Space

Address	Memory	
N	0x2B	"+"
N+1	0x36	"6"
N+2	0x35	"5"
N+3	0x20	SP

b

BEL

Address	Memory	
N	0x42	"B"
N+1	0x45	"E"
N+2	0x4C	"L"
N+3	0x07	BEL

c

Chapter 2

Data Organisation in Memory

Array, String and Structure Representations

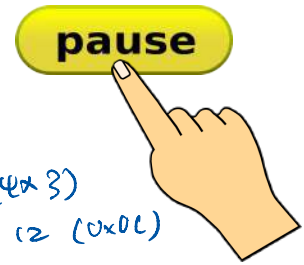
Learning Objectives (2.4)
1. Describe the representation of arrays in memory.
2. Describe the C and Pascal strings storage in memory.
3. Describe the representation of structures in memory

long int = 4 bytes

Quiz: Accessing Array Elements

- What is the start address of the array variable `A[3]` if the base address of the array `A` is hexadecimal `0x100`?

```
        :
long int A[5];
A[3] = 1;
```



BA = 0x100
offset = 0x100 + (4x3)
= 0x100 + 12 (0x0C)
= 0x10C

0x112	0x10C	0x103	0x100
a	b	c	d

Quiz: Accessing Array Elements

- What is the start address of the array variable `A[3]` if the base address of the array `A` is hexadecimal `0x100`?

```
        :
long int A[5];
A[3] = 1;
```

Since `long int` is 4 bytes, the start addresses of `A[3]` should be:
BA+(3x4) = `0x100` + `0x00C` = `0x10C`
Note: 12 bytes from `0x100` is `0x10C` (not `0x112`)
(addresses use concise hexadecimal notation)

0x112	0x10C	0x103	0x100
a	b	c	d

Chapter 2

Data Organisation in Memory

Pointer Representation

Learning Objectives (2.5)
1. Describe the representation of pointers in memory.

Quiz: Memory Allocation

- A C compiler is compiling for a CPU with a 32-bit address bus. How many bytes in memory will it allocate for the variable declarations shown?

```
        :
short int I;
short int *P;
```

pause

1 byte = 8 bits
32bit = 4 bytes
*p = 4 bytes
short int = 2 bytes
6 Bytes

2 bytes (a) 4 bytes (b) 6 bytes (c) 8 bytes (d)

CZ1106 CE1106

Quiz: Memory Allocation

- A C compiler is compiling for a CPU with a 32-bit address bus. How many bytes in memory will it allocate for the variable declarations shown?

```
        :
short int I;   2 bytes
short int *P;  4 bytes
```

The **short int** data type is 2 bytes. A pointer data type represents an address and will require a memory size that matches the number of bits needed to specify a unique address (i.e. 32 bits or 4 bytes). Therefore the total memory allocated by the C compiler will be **2 + 4**, which is **6** bytes.

2 bytes (a) 4 bytes (b) **6 bytes (c)** 8 bytes (d)

CZ1106 CE1106

Chapter 2

Data Organisation in Memory

Data Alignment

Learning Objectives (2.6)

1. Describe the concept of data alignment.
2. Describe data alignment considerations for efficient access and storage of structure variables in memory.

CZ1106 CE1106

Quiz: Data Alignment

- How many bytes will be allocated for variables `S1` and `S2`? Assume multi-byte data requires even address data alignment and memory allocation starts at address `0x000`.

```
struct rec {
  char C;        → 1 byte
  short int X;   → 2 bytes
 }
 rec S1,S2;
```

pause

2 bytes (a) 4 bytes (b) 6 bytes (c) 8 bytes (d)

CZ1106
CE1106

Quiz: Data Alignment

- How many bytes will be allocated for variables `S1` and `S2`? Assume multi-byte data requires even address data alignment and memory allocation starts at address `0x000`.

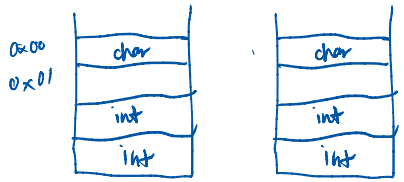
```
struct rec {
  char C;
  short int X;
 }
 rec S1,S2;
```

Character `C` of `S1` occupies byte at `0x000`. But next available memory location `0x001` is an odd address, a padded byte will be added to ensure the two-byte variable `X` starts at as even address `0x002`. Structure `S1` takes 4 bytes. The same is repeated for `S2` starting at `0x004`. Total allocation is 8 bytes.

2 bytes **4 bytes** **6 bytes** **8 bytes**

a b c C

Must start at even address



S1 S2

Total: 8 bytes

Chapter 3

Instruction Organisation in Memory

Quiz Questions in Video Lecture

CZ1106
CE1106

Chapter 3

Instruction Organisation in Memory

Characteristics of Instructions and the ARM Programmer's Model

Learning Objectives (3.1)

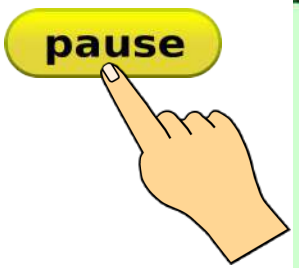
1. Describe the characteristics and role of an instruction.
2. Describe the function of the ARM instruction set **MOV** instruction
3. Describe the ARM programmer's model.
4. Describe the functions and interpretation of several ARM registers.

CZ1106
CE1106

Quiz: The MOV Instruction

- Based on the initial state shown, which option is the correct register state after executing the ARM mnemonic `MOV R3,R4`?

R3 0xAABBCCDD
R4 0x00000000
Before execution



a	b	c	d
R3 0x00000000	R3 0xAABBCCDD	R3 0xAABBCCDD	R3 0x00000000
R4 0x00000000	R4 0xAABBCCDD	R4 0x00000000	R4 0xAABBCCDD
After execution	After execution	After execution	After execution

CZ1106
CE1106

Quiz: The MOV Instruction

- Based on the initial state shown, which option is the correct register state after executing the ARM mnemonic `MOV R3,R4`?

R3 0xAABBCCDD
R4 0x00000000
Before execution

Destination ← `R3` , `R4` → Source

a	b	c	d
R3 0x00000000	R3 0xAABBCCDD	R3 0xAABBCCDD	R3 0x00000000
R4 0x00000000	R4 0xAABBCCDD	R4 0x00000000	R4 0xAABBCCDD
After execution	After execution	After execution	After execution

Quiz: ARM Programmer's Model

- Which of the following is(are) **not** valid register(s) in the ARM programmer's model?
  Note: You can indicate more than one selection.

pause

R16 PC LR V SP

a b c d e

CZ1106
CE1106

Quiz: ARM Programmer's Model

- Which of the following is(are) **not** valid register(s) in the ARM programmer's model?
  Note: You can indicate more than one selection.

ARM has 16 32-bit registers, **R0** to **R15**. As such, **R16** is not a valid register. **R13**, **R14** and **R15** are also known as the **SP**, **LR** and **PC** respectively. The overflow **V** flag is not a register but a condition code bit (or flag) in the CPSR. It indicates if an overflow has occurred during a signed arithmetic operation.

R16 PC LR V SP

a b c d e

CZ1106
CE1106

Chapter 3

Instruction Organisation in Memory

Basic Execution Cycle

Learning Objectives (3.2)

1. Describe the function of a simple **LDR** instruction from the ARM instruction set.
2. Describe the basic execution cycle of a simple **LDR** instruction.
3. Describe the factor that determines the execution speed of an instruction.

CZ1106
CE1106

Quiz: Instructions & Memory Access

- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.

pause

a
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA) (1 MA)

b
```
MOV R1,R0
LDR R1,[R0]
```
(2 MA) (2 MA)

c
```
MOV R1,R0
LDR R1,[R0]
```
(2 MA) (1 MA)

d
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA) (2 MA)

CZ1106 CE1106

Quiz: Instructions & Memory Access

- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.

a
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(1 MA)

b
```
MOV R1,R0
LDR R1,[R0]
```
(2 MA)
(2 MA)

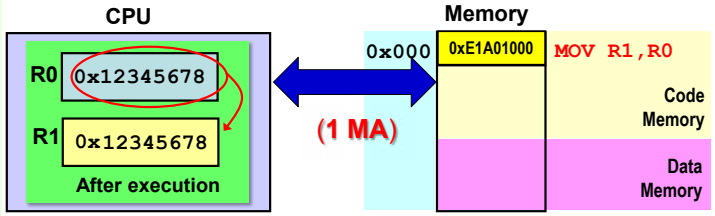
c
```
MOV R1,R0
LDR R1,[R0]
```
(2 MA)
(1 MA)

d
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(2 MA)

CZ1106 CE1106

Quiz: Instructions & Memory Access

- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.

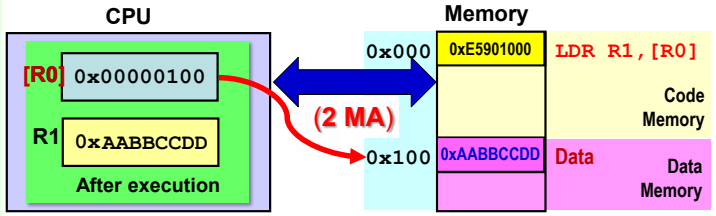


d
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(2 MA)

CZ1106 CE1106

Quiz: Instructions & Memory Access

- Select the option that correctly describes the number of memory access (**MA**) required to execute the two ARM instructions shown.



STR also need 2 MA

d
```
MOV R1,R0
LDR R1,[R0]
```
(1 MA)
(2 MA)

Chapter 4

Addressing Modes

Quiz Question for Video Lectures

CZ1106
CE1106

Chapter 4

Addressing Modes

Introduction to Assembly Programming and Addressing Modes

Learning Objectives (4.1)

1. Identify why and when to use assembly language programming.
2. Describe what are addressing modes.

CZ1106
CE1106

Chapter 4

Addressing Modes

Register Direct and Immediate Addressing

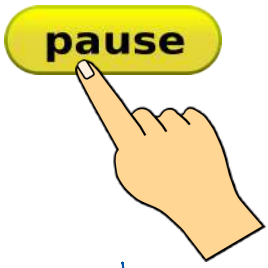
Learning Objectives (4.2)

1. Describe what is register direct.
2. Describe what is immediate data and its application.

CZ1106
CE1106

Quiz: The MOV mnemonic

- Which of the following MOV mnemonic(s) is(are) **not valid** ARM instructions?
- You may select more than one answer.



`MOV R1,R13` a

`MOV R1,SP` b

`MOV #0,R0` c

`MOV R2,CPSR` d

Quiz: The MOV mnemonic

- Which of the following MOV mnemonic(s) is(are) **not valid** ARM instructions?
- You may select more than one answer.

Register R13 is also known as the stack pointer (SP). Both notations are valid for register direct. ✓

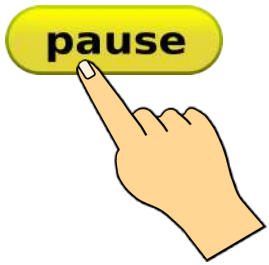
Immediate values can only be used as source operands. ✗

Current Processor Status Register (CPSR) cannot be access using the MOV operator. ✗

`MOV R1,R13` (a) `MOV R1,SP` (b) `MOV #0,R0` (c) `MOV R2,CPSR` (d)

Quiz: Immediate Addressing

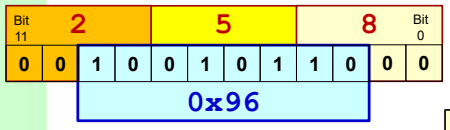
- Which of the following immediate value(s) of I is(are) **not valid** for the ARM mnemonic `MOV R3,#I`?
- Answers are given in both decimal and hexadecimal notations.
- You may select more than one answer.

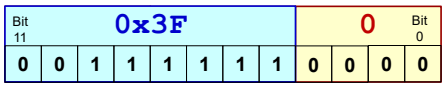


I = 256 (a) I = 258 (b) I = 0x258 (c) I = 0x3F0 (d)

Quiz: Immediate Addressing

- Which of the following immediate value(s) of I is(are) **not valid** for the ARM mnemonic `MOV R3,#I`?





Decimal value of 0x100 ✓ — I = 256 (a)

Decimal value of 0x102 ✗ — I = 258 (b)

8-bit hex value of 0x96 rotated right by 30 (*n*=15) bits ✓ — I = 0x258 (c)

8-bit hex value of 0x3F rotated right by 28 (*n*=14) bits ✓ — I = 0x3F0 (d)

Chapter 4

Addressing Modes

Register Indirect with Base Register

Learning Objectives (4.3)

1. Describe what is register indirect and the ARM instructions that support this addressing mode.
2. Describe the variants and application of register indirect that uses base plus offset and index register.
3. Compare the relative pros and cons of register direct and register indirect addressing modes.

CZ1106 CE1106

Quiz: Register Indirect

- Given the initial states shown, which are the correct states of `R0` and `R2` after the execution of the instruction `STR R2,[R0,#4]`?

pause

Initial States

R0	0x00000100
R2	0x12345678

Address	Memory
0x100	0x00
0x101	0x00
0x102	0x00
0x103	0x00
0x104	0x88
0x105	0x88
0x106	0x88
0x107	0x88
:	:

a

R0	0x00000100
R2	0x12345678

b

R0	0x00000104
R2	0x88888888

c

R0	0x00000100
R2	0x88888888

d

R0	0x00000100
R2	0x00000000

CZ1106 CE1106

Quiz: Register Indirect

- Given the initial states shown, which are the correct states of `R0` and `R2` after the execution of the instruction `STR R2,[R0,#4]`?

Initial States

R0	0x00000100
R2	0x12345678

Address	Memory
0x100	0x00
0x101	0x00
0x102	0x00
0x103	0x00
0x104	0x88
0x105	0x88
0x106	0x88
0x107	0x88
:	:

a

R0	0x00000100
R2	0x12345678

b

R0	0x00000104
R2	0x1245678

c

R0	0x00000100
R2	0x88888888

d

R0	0x00000100
R2	0x00000000

no exclamation mark ∴ not recommended

Adding `R0` with offset `4` to get the effective address of memory operand does not alter content of indirect register `R0` after execution.

`STR` operator copies register content to memory. So source register is not altered after execution. Only destination memory location is modified.

CZ1106 CE1106

Quiz: Execution Speed

- With the initial states shown, which instruction will be **able** to load the value `0x102` into register `R2` with the **least** number of clock cycles?

pause

Initial States

R0	0x00000102
R2	0x12345678

Address	Memory
0x100	0x02
0x101	0x01
0x102	0x00
0x103	0x00
0x104	0x00
0x105	0x00
:	:

a `MOV R2,#0x102` (invalid)

b `MOV R2,R0`

c `LDR R2,[R0]`

d `LDR R2,[R0,#-2]`

Note: Multi-byte data storage in memory is in **Little Endian** format.

CZ1106 CE1106

Quiz: Execution Speed

- With the initial states shown, which instruction will be **able** to load the value `0x102` into register `R2` with the **least** number of clock cycles?

Initial States

R0	0x00000102
R2	0x12345678

Address	Memory
0x100	0x02
0x101	0x01
0x102	0x00
0x103	0x00
0x104	0x00
0x105	0x00
:	:

`MOV R2,#0x102` ✗ The immediate value `#0x102` is not valid.

`MOV R2,R0` ✓ `0x102` in `R0` is copied into `R2`. This takes only 1 memory cycle.

`LDR R2,[R0]` ✗ Address `0x102` in `R0` does not meet data alignment constraints for 4-byte memory access.

`LDR R2,[R0,#-2]` ✗ Effective address `0x100` meets data alignment constraints but `LDR` takes 2 memory cycles.

CZ1106 CE1106

Chapter 4

Addressing Modes

Register Indirect with Autoindexing and Stacks

Learning Objectives (4.4)

1. Describe what is autoindexing feature of ARM's register indirect addressing mode.
2. Describe the differences between pre-index and post-index addressing modes.
3. Describe the various stack implementations and operations using the ARM addressing modes.

CZ1106 CE1106

Quiz: Autoindexing

- Given the initial states shown, which are the correct states of `R0` and `R2` after the execution of the instruction `LDR R2,[R0],#4`?

pause

Initial States

R0	0x00000100
R2	0x12345678

Address	Memory
0x100	0x00
0x101	0x00
0x102	0x00
0x103	0x00
0x104	0x88
0x105	0x88
0x106	0x88
0x107	0x88
:	:

a. R0 0x00000100, R2 0x00000000

b. R0 0x00000104, R2 0x88888888

c. R0 0x00000100, R2 0x88888888

d. R0 0x00000104, R2 0x00000000

CZ1106 CE1106

Quiz: Autoindexing

- Given the initial states shown, which are the correct states of `R0` and `R2` after the execution of the instruction `LDR R2,[R0],#4`?

Initial States

R0	0x00000100
R2	0x12345678

Address	Memory
0x100	0x00
0x101	0x00
0x102	0x00
0x103	0x00
0x104	0x88
0x105	0x88
0x106	0x88
0x107	0x88
:	:

✗ a. R0 0x00000100, R2 0x00000000

Autoindex will modify the indirect register `R0`.

✗ b. R0 0x00000104, R2 0x88888888

Offset with **pre-index** autoindexing given by `LDR R2,[R0,#4]!` gives this answer.

✗ c. R0 0x00000100, R2 0x88888888

This is offset with **post-index** autoindexing. This mean `EA` is given by `0x100` in `R0` before offset `4` is added to `R0`.

✓ d. R0 0x00000104, R2 0x00000000

CZ1106 CE1106

Push – `STR R0,[SP,#-4]!`
Pop – `LDR R0,[SP],#4`

✓A. Full Descending (FD)

CZ1106 CE1106

Quiz: The ED Stack

- Give the mnemonic to **push** the register `R1` on to an **Empty Descending** (ED) stack.

0xFE000003	
SP → 0xFE000004	
0xFE000005	
0xFE000006	
0xFE000007	
0xFE000008	0x44
0xFE000009	0x33
0xFE00000A	0x22
0xFE00000B	0x11
0xFE00000C	

pause

Full ascending

a. `STR R1,[R13,#4]`

Empty descending

b. `STR R1,[R13],#-4`

c. `STR R1,[R13,#-4]!`

d. `STR R1,[R13],#4`

Full descending

Empty ascending

Quiz: The ED Stack

- Give the mnemonic to **push** the register `R1` on to an **Empty Descending** (ED) stack.

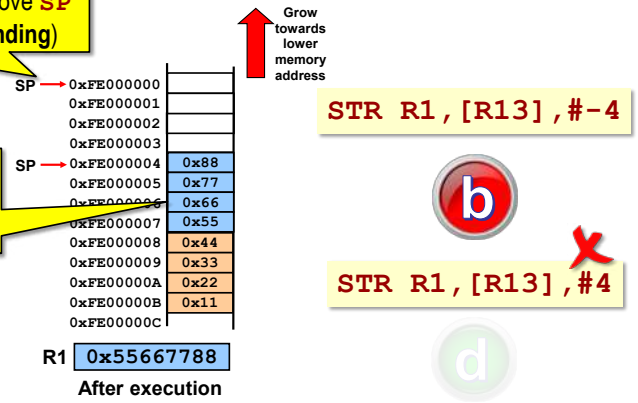
Then, add **-ve** offset of `-4` to `R13` to move `SP` towards lower memory address (**descending**)

`STR R1,[R13,#4]` ✗

First, copy `R1` to the empty stack space `SP` is currently pointing to before autoindexing. Need **post-index**.

`STR R1,[R13,#-4]!` ✗

c



`STR R1,[R13],#-4`

b

`STR R1,[R13],#4` ✗

d

Chapter 4

Addressing Modes

PC-related Addressing Modes

Learning Objectives (4.5)

1. Describe difference between absolute & relative jump.
2. Describe the concept of position-independent code and how it is achieved.
3. Describe how data can be accessed using PC relative addressing

Quiz: Position-Independent

- Select the instructions below that can be used at label `Q1`, do what is in the comments and are **position-independent (P-I)**.
- Note: You may select more than one answer.

Address	ARM code segment
0x00	MOV R0,#0x20
0x04	Q1 ???
:	:
0x20	Jmp MOV R0,R2
:	:

pause

`MOV R1,R0` ;copy R0 to R1

a

`B Jmp` ;jump to Jmp

b

`MOV PC,#0x20` ;jump to Jmp

c ✗

`MOV PC,R0` ;jump to Jmp

d ✗

Quiz: Position-Independent

- Select the instructions below that can be used at label `Q1`, do what is in the comments and are **position-independent (P-I)**.

Address	ARM code segment
0x00	MOV R0,#0x20
0x04	Q1 ???
:	:
0x20	Jmp MOV R0,R2
:	:

✓ `MOV R1,R0` ;copy R0 to R1

Changing any register other than the **PC** is always P-I.

✓ `B Jmp` ;jump to Jmp

The ARM branch instruction `B` used a **PC** relative offset to carry out the jump and is therefore P-I.

`MOV PC,#0x20` ;jump to Jmp ✗

`MOV PC,R0` ;jump to Jmp ✗

Loading **PC** with the absolute start address (i.e. `0x20`) of the instruction to jump to violates the P-I requirement that jump must be relative to the current **PC** value. It does not matter whether immediate or register direct is used to modify the **PC**.

Chapter 5

Instruction Set

Quiz Questions for Video Lectures

CZ1106
CE1106

Chapter 5

Instruction Set

Data Transfer Instructions

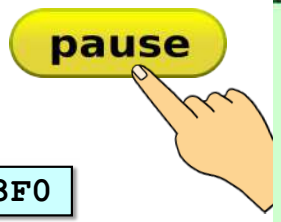
Learning Objectives (5.1)

1. Describe how data in register and memory can be efficiently transferred.
2. Describe how byte-sized data can be access in memory.

CZ1106
CE1106

Quiz: The MVN Instruction

- Given the 32-bit hexadecimal value in `R0` after executing the instruction `MVN R0,#0x3F0`.



Invalid mnemonic (a)

R0 0x000003F0 (b)

R0 0xFFFFFC0F (c)

R0 0x00000C0F (d)

CZ1106
CE1106

Quiz: The MVN Instruction

- Given the 32-bit hexadecimal value in `R0` after executing the instruction `MVN R0,#0x3F0`.

`MVN` will complement the 32-bit immediate value before moving it to the destination register.

Invalid mnemonic (a)

R0 0x000003F0 (b)

R0 0xFFFFFC0F (c)

R0 0x00000C0F (d)

32-bit value of `0x3F0` is `0x000003F0`.
Its complement is `0xFFFFFC0F`.
Reminder:
$3 = 0011_2$
Not $(3) = 1100_2$
$= 0xC$

CZ1106 CE1106

Quiz: Data Transfer Instructions

- Based on the initial states, which instruction(s) (after execution) will give this result:
  Note: You may select more than one answer.

After Execution

R0	0x00000100
R2	0x000000F8

pause

Initial States

R0	0x00000100
R2	0xFFFFFF07

Address	Memory
0x100	0xF8
0x101	0x00
0x102	0x00
0x103	0x00
0x104	0x00
0x105	0x00
0x106	0x00
0x107	0xF8
:	:

`MVN R2,#0x7` — a — 0xFFFFFFF8

`LDRB R2,[R0,#7]` — b

`MVN R2,R2` — c

`LDR R2,[R0],#4` — d

Note: Multi-byte data storage in memory is in **Little Endian** format.

CZ1106 CE1106

Quiz: Data Transfer Instructions

- Based on the initial states, which instruction(s) (after execution) will give this result:
  Note: You may select more than one answer.

After Execution

R0	0x00000100
R2	0x000000F8

Initial States

R0	0x00000100
R2	0xFFFFFF07

Address	Memory
0x100	0xF8
0x101	0x00
0x102	0x00
0x103	0x00
0x104	0x00
0x105	0x00
0x106	0x00
0x107	0xF8
:	:

`MVN R2,#0x7` ✗ — The 32-bit complement of `0x7` copied into `R2` is `0xFFFFFFF8`.

`LDRB R2,[R0,#7]` ✓ — EA is `0x107`, so `0xF8` zero-extends in `R2`. Byte access - no need data alignment.

`MVN R2,R2` ✓ — `0xFFFFFF07` in `R2` when complemented will become `0x000000F8`. This value moves back into itself in `R2`.

`LDR R2,[R0],#4` ✗ — The register indirect with post-index will load `0x000000F8` into `R2` but `R0` will be changed to `0x104` due to autoindexing.

©2020, SCSE/NTU

CZ1106 CE1106

Chapter 5

Instruction Set

Arithmetic Instructions

Learning Objectives (5.2)

1. Describe the operation and uses of the basic arithmetic instructions in the ARM instruction set.
2. Describe how arithmetic operations influence the status of Condition Code flags.

©2020, SCSE/NTU

CZ1106 CE1106

Quiz: Binary Subtraction

- What is the result of the 5-bit binary subtraction 01001_2 - 11100_2 if both these numbers are signed numbers?

pause

a — 00101_2

b — 00011_2

c — 01101_2

d — 10010_2

©2020, SCSE/NTU

CZ1106 CE1106

Quiz: Binary Subtraction

- What is the result of the 5-bit binary subtraction 01001_2 - 11100_2 if both these numbers are signed numbers?

11100
00011
+ 1

a 00101_2

b 00011_2

c 01101_2

d 10010_2

Subtraction	Decimal value	Addition equivalent	Decimal value
01001	(+9)	01001	(+9)
-11100	- (-4)	+00100	+ (+4)
	(13)	01101	(13)

Negate using two's complement operation

$01001_2 - 11100_2 = 01101_2$
Subtraction can be done using addition. The subtrahend is negated by using 2's complement. Then add the 2 numbers.

CZ1106 CE1106

Quiz: The ADD Instruction

- What would be the value in register R0 after the ADD instruction shown in executed **10** times.

```
        MOV   R0,#1
Loop    ADD   R0,R0,R0
```

loop back 9 times

pause

a R0 0x00000001

b R0 0x00000008

c R0 0x00000100

d R0 0x00000256

CZ1106 CE1106

Quiz: The ADD Instruction

- What would be the value in register R0 after the ADD instruction shown in executed **8 times**.

```
        MOV   R0,#1
Loop    ADD   R0,R0,R0
```

loop back 7 times

R0=R0+R0

Iteration *n*:
1. R0 = 1 + 1 = 2
2. R0 = 2 + 2 = 4
3. R0 = 4 + 4 = 8

n. R0 = 2^n

8. R0 = 2^8 = (256)

R0=0x00000100

a R0 0x00000001 ✗

b R0 0x00000008 ✗

c R0 0x00000100 ✓

d R0 0x00000256 ✗

256 in decimal

CZ1106 CE1106

Chapter 5

Instruction Set

Logical, Shift and Rotate Instructions

Learning Objectives (5.3)

1. Describe the operation and uses of the various logical instructions.
2. Describe the operation and uses of the various shift and rotate instructions.
3. Describe how multiplication and division can be done using bit shift.

CZ1106 CE1106

Quiz: Logical and Shift Instructions

- Given the initial states shown, which is the correct state of `R2` after executing the instruction `EOR R2,R0,R1,ASR R3`?

pause

Initial States

R0	0x11223344
R1	0xF0000000
R2	0x12345678
R3	0x00000004

a. R2 0x12345678

b. R2 0x11223344

c. R2 0xEE223344

d. R2 0x1E223344

CZ1106 CE1106

Quiz: Logical and Shift Instructions

- Given the initial states shown, which is the correct state of `R2` after executing the instruction `EOR R2,R0,R1,ASR R3`?

Initial States

R0	0x11223344
R1	0xF0000000
R2	0x12345678
R3	0x00000004

`R2=R0 EOR (0xFF000000)`
`R2=0xEE223344`
Flip 8 MSB

`=(R1 >>sign R3)`
`=(R1 >>sign 4)`
`=(0xF0000000 >>sign 4)`
`=(0xFF000000)`

a. R2 0x12345678

b. R2 0x11223344

c. R2 0xEE223344 ✓

d. R2 0x1E223344

CZ1106 CE1106

Quiz: An ARM Code Segment

- `R0` contains a zero-extended byte read from memory. What would be the value in `R0` after executing the code segment shown?

pause

R0 0x00000080

```
MOV R1,#0
SUB R2,R1,R0,LSR #7
ORR R0,R0,R2,LSL #8
```

a. R0 0x00000000

b. R0 0x00000080

c. R0 0xFFFFFFFF

d. R0 0xFFFFFF80

CZ1106 CE1106

Quiz: An ARM Code Segment

- `R0` contains a zero-extended byte read from memory. What would be the value in `R0` after executing the code segment shown?

R0 0x00000080

```
MOV R1,#0
SUB R2,R1,R0,LSR #7
ORR R0,R0,R2,LSL #8
```

R0 ..00000000001

`R0=R0 OR (R2<<8)`
`R0=R0 OR 0xFFFFFF00`
`R2=0xFFFFFF80`

`=(R2<<8)`
`=0xFFFFFF00`

`R2=R1-(R0>>7)`
`R2= 0-(R0>>7)`
`R2= 0-(1)`
`R2= 0xFFFFFFFF`

This code sign-extends the zero-extended byte fetched from memory to 32-bits

a. R0 0x00000000

b. R0 0x00000080

c. R0 0xFFFFFFFF

d. R0 0xFFFFFF80 ✓

CZ1106
CE1106

Chapter 5

Instruction Set

Program Control Instructions

Learning Objectives (5.4)

1. Describe the various conditional branch instructions and its uses.
2. Describe how conditional test can be implemented.

CZ1106
CE1106

Quiz: Loop Count

- How many times will the `BMI Loop` instruction be executed in the code segment shown?

```
      MOV   R1,#-2
Loop  ADDS  R1,R1,#1
      BMI   Loop
```

R1 = -2
R1 = -1
R1 = 0
N = 1

pause

0 a 1 b 2 c 3 d

CZ1106
CE1106

Quiz: Loop Count

- How many times will the `BMI Loop` instruction be executed in the code segment shown?

```
      MOV   R1,#-2
Loop  ADDS  R1,R1,#1
      BMI   Loop
```

N=1 (loop back)
N=0 (exit)

R1 = -2 → N = 1
R1 = -1 → N = 1
R1 = 0 → N = 0

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned
CC or LO	C = 0	Lower, unsigned
MI	N = 1	Negative
PL	N = 0	Positive or zero

0 a 1 b 2 c 3 d

Execute twice. Once branch taken and once branch not taken.

CZ1106
CE1106

Quiz: The CMP Instruction

- Which of the following values in `R0` and `R1` will cause the `MOV R2,R3` instruction to be executed?
- **Note:** You may choose more than one correct answer

```
      CMP  R0,R1
      BHS  Skip
      MOV  R2,R3
Skip  :
```

Higher or same, C = 1

pause

a. R0 = 0xABCDEFFF, R1 = 0xABCDEFFF

b. R0 = 0x00000001, R1 = 0x00000005

c. R0 = 0x00000005, R1 = 0x00000001

d. R0 = 0x00000005, R1 = 0xFFFFFFFF

Quiz: The CMP Instruction

- Which of the following values in `R0` and `R1` will cause the `MOV R2,R3` instruction to be executed?

```
CMP R0,R1
BHS Skip
MOV R2,R3
Skip :
```

`BHS` test unsigned numbers. If unsigned value in (`R0 ≥ R1`), `BHS` will skip over the `MOV` instruction

`R0≥R1` → `Skip :` `R0<R1`

To execute `MOV R2,R3` (`R0 < R1`) must be true.

Suffix	Flags	Meaning
EQ	Z = 1	Equal
NE	Z = 0	Not equal
CS or HS	C = 1	Higher or same, unsigned
CC or LO	C = 0	Lower, unsigned

a. `R0=R1` ✗
R0 0xABCDEFFF
R1 0xABCDEFFF

b. ✓ `R0<R1`
R0 0x00000001
R1 0x00000005

c. `R0≥R1` ✗
R0 0x00000005
R1 0x00000001

d. ✓ `R0<R1`
R0 0x00000005
R1 0xFFFFFFFF

Chapter 5

Program Example

Finding the Largest Number

Learning Objectives (5.5)

1. Use appropriate data transfer instructions to retrieve memory arrays efficiently.
2. Use appropriate program control instructions to determine flow of program based on desired outcomes.
3. Implement a simple find max algorithm in ARM assembly.

Quiz: Code Optimisation

- Which mnemonic(s) in **FindMax** can you remove to optimised the program?
- Note: You may select more than one instructions.

pause

```
        MOV   R0,#0x100     ;setup pointer to first array element
        MOV   R1,#9         ;load 9 into counter register
        LDR   R3,[R0]   (a) ;assume 1st no. in array is current max
Loop    ADD   R0,R0,#4  (b) ;increment array pointer to next element
        LDR   R2,[R0]       ;get next no. in array
        CMP   R2,R3         ;compare R3 and R2 (i.e. R2-R3)
        BLS   Skip      (c) ;branch if R2 <  current max (i.e. R3)
        MOV   R3,R2         ;update current max. in R3 with R2
Skip    SUBS  R1,R1,#1  (d) ;decrement 1 from counter register
        BNE   Loop          ;jump back to Loop if not zero
```


Quiz: Code Optimisation

- Which mnemonic in **FindMax** can you remove by using a more optimised addressing mode?

```
        MOV   R0,#0x100     ;setup pointer to first array element
        MOV   R1,#9         ;load 9 into counter register
        LDR   R3,[R0]   (a) ;assume 1st no. in array is current max
Loop    LDR   R2,[R0,#4]! (b) ;pre-index autoindex to next element
                            ;then get array element from memory
        CMP   R2,R3         ;compare R3 and R2 (i.e. R2-R3)
        BLS   Skip      (c) ;branch if R2 <  current max (i.e. R3)
        MOV   R3,R2         ;update current max. in R3 with R2
Skip    SUBS  R1,R1,#1  (d) ;decrement 1 from counter register
        BNE   Loop          ;jump back to Loop if not zero
```

Change saves 9 cycles.

CZ1106
CE1106

Conditional Execution

- ARM instructions can be conditionally executed based on the CC flags.

```
; C code
if (R0 == 1)
 R1 = 3;
else
 R1 = 5;
```

ARM code example

```
      CMP   R0,#1          ; set CC based on r0 –1
      BNE   ELSE           ; if (R0 == 1)
      MOV   R1,#3          ; then { R1 := 3}
      B     SKIP           ; skip over else code seg
ELSE  MOV   R1,#5          ; else  { R1 := 5}
SKIP      .....            ;
```

- The conditional execution feature allows us to make the execution of each instruction dependent on the current status of the **N**, **Z**, **V**, **C** flags.

```
      CMP   R0,#1          ; if (r0 == 1)
      MOVEQ R1,#3          ; then { r1 := 3}
      MOVNE R1,#5          ; else  { r1 := 5}
SKIP  .....                ;
```

Executes if **Z** = 1

Executes if **Z** = 0

CZ1106
CE1106

Quiz: Code Optimisation

- Which mnemonic in **FindMax** can you remove by using a more optimised addressing mode?

```
      MOV   R0,#0x100    ;setup pointer to first array element
      MOV   R1,#9        ;load 9 into counter register
      LDR   R3,[R0]      ;assume 1st no. in array is current max

Loop  LDR   R2,[R0,#4]!  ;pre-index autoindex to next element
                         ;then get array element from memory
      CMP   R2,R3        ;compare R3 and R2 (i.e. R2-R3)
                         ;BLS removed with conditional execution
      MOVHI R3,R2        ;current max is update with R2 if R2>R3

Skip  SUBS  R1,R1,#1     ;decrement 1 from counter register
      BNE   Loop         ;jump back to Loop if not zero
```

a b c d

1 byte = 8 bits

1 MB = 1,048,576

decimal value of 8-bit hex 2's complement 0x80?

0x80 = 1000 0000

$-(0111\ 1111)$

$-(1000\ 0000)$

$= -128$

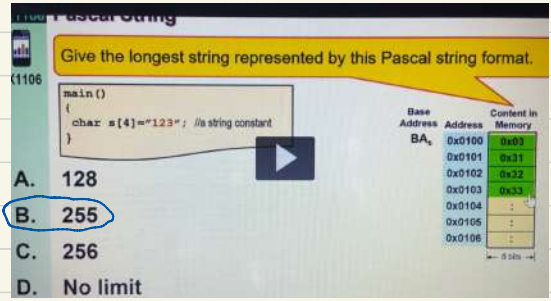
Find a positive short integer in the memory map if Little Endian byte-ordering is used.

0xF3
0x 72
0x82

82 72 F3 Ans: 0x 72 F3

short int occupies 2 bytes

Pascal string format



largest byte-sized value for 8-bits is 255

Mock Quiz

0x 00
0x 36
0x 00

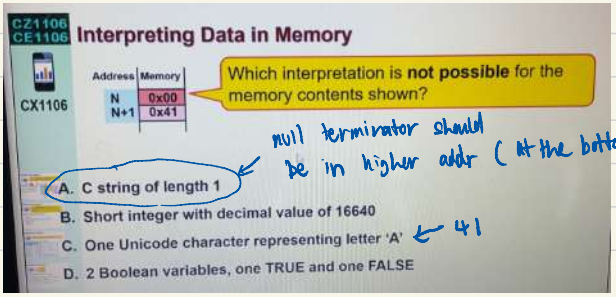
POSSIBLE interpretations:

→ Unicode "6"

→ 3 bool values (2 false, 1 true)

→ C string of length 1

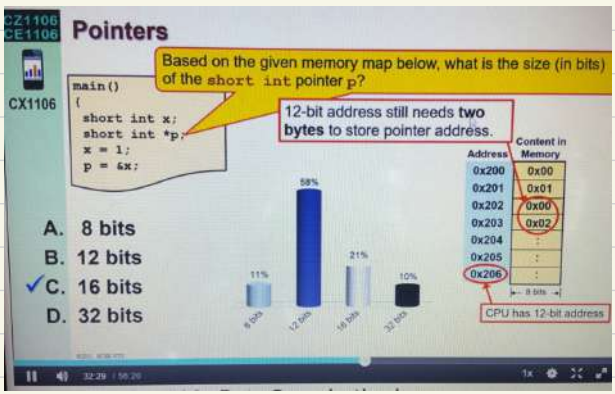
Interpretation of memory content



null terminator should be in higher addr (at the bottom)

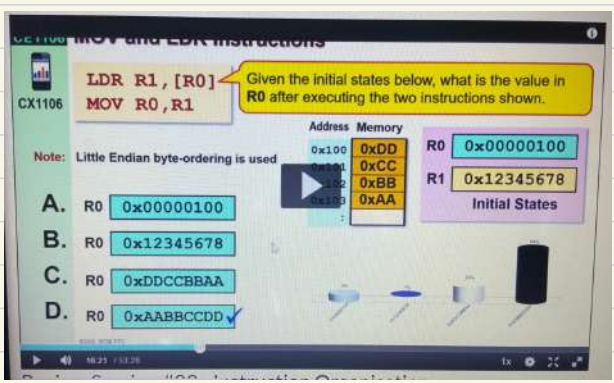
41

Pointer p



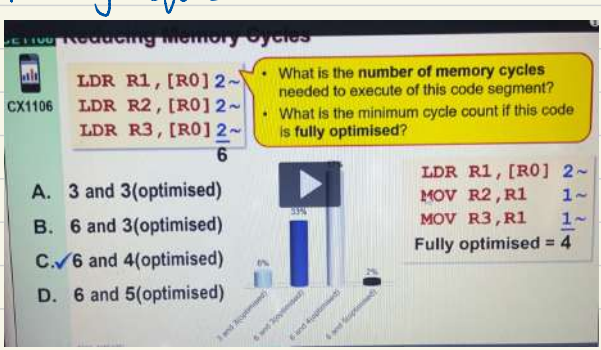
only possible to store 8, 16, 32, 64 bits

MOV and LDR instructions

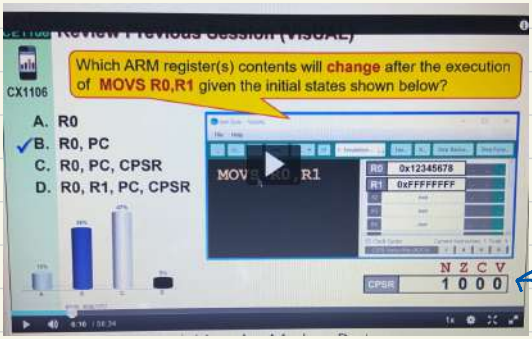


R1 = AABBCCDD

memory cycles

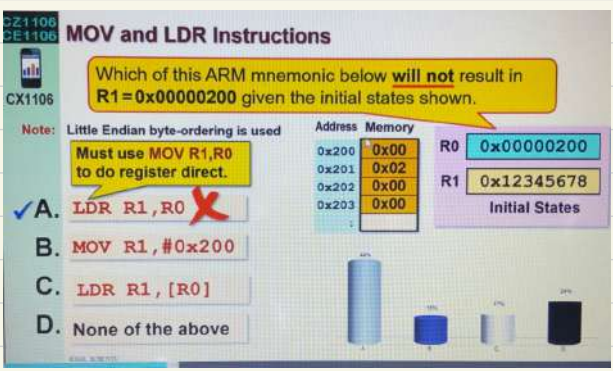


Change in ARM Content

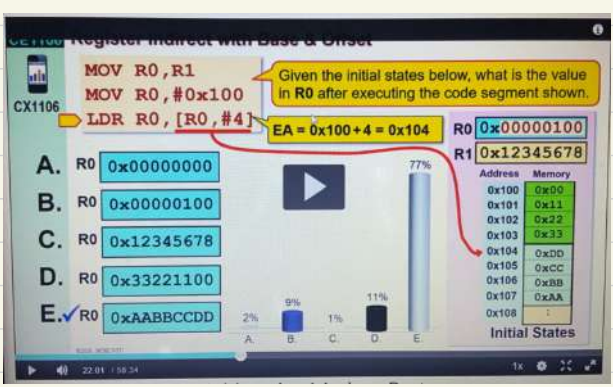


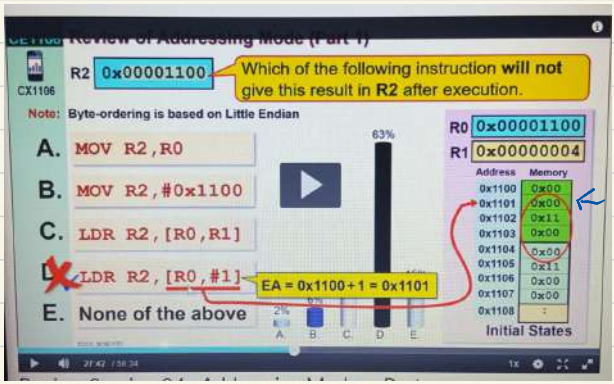
← N and set

MOV and LDR instructions



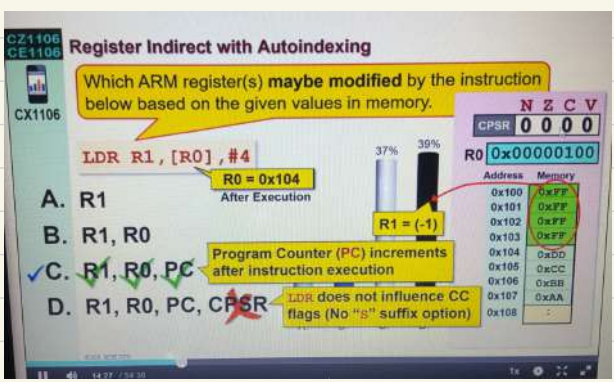
Register Indirect with Base & Offset





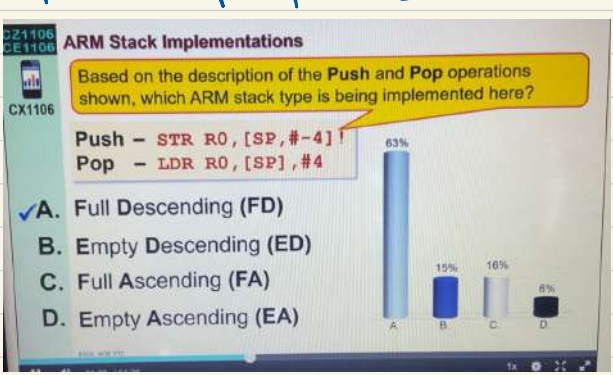
data alignment violation

Register Indirect with auto indexing



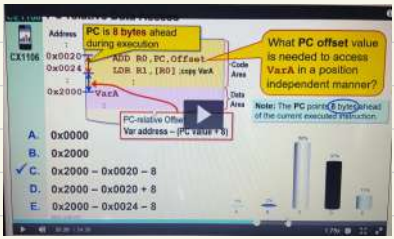
No such thing as LDRS

Push & Pop operations

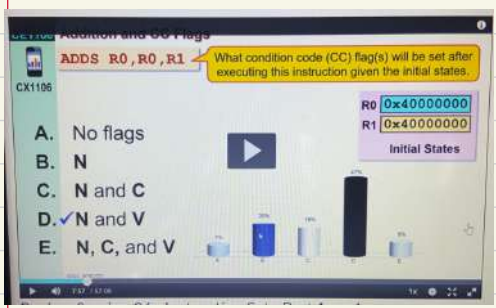


Full – SP point to top of stack (occupied space)

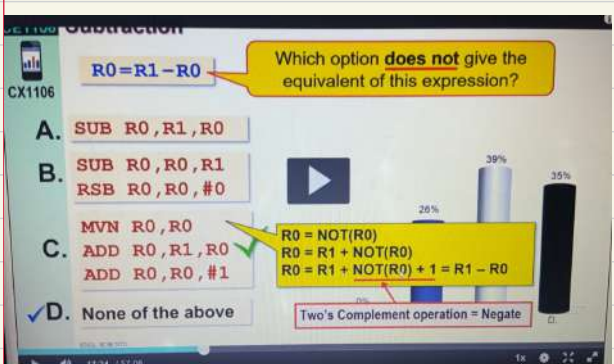
PC-relative Data Access



ADDS



Sub

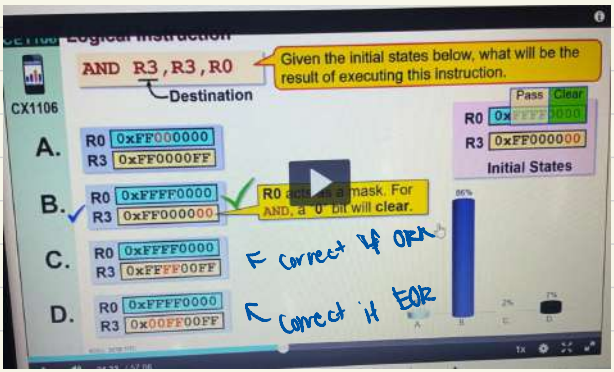


B. $R0 = R0 - R1$

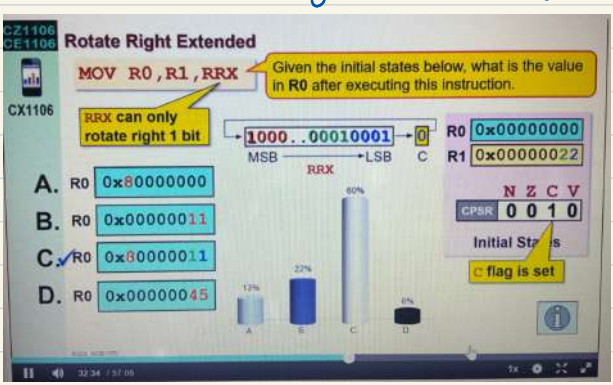
$R0 = 0 - (R0 - R1)$

$= R1 - R0$

Logical instruction AND



RRX (Rotate Right Extended)

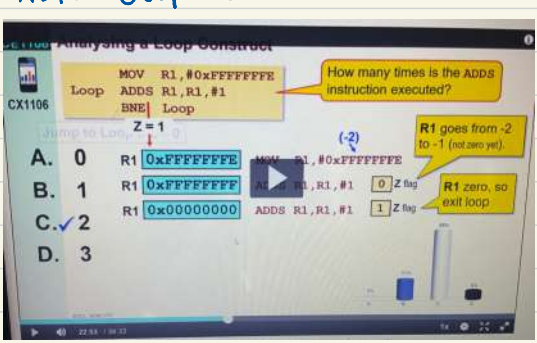


C flag extends

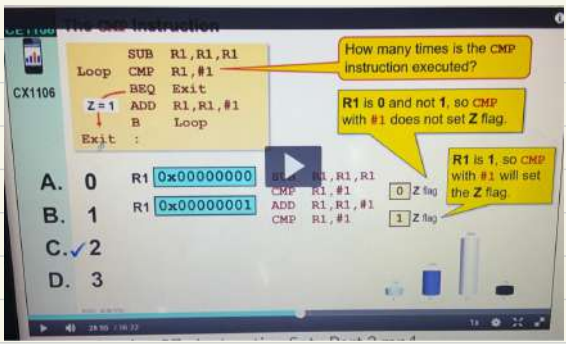
0010 0010 → shift it
0001 0001

no change in CPSR as MOV has NO S

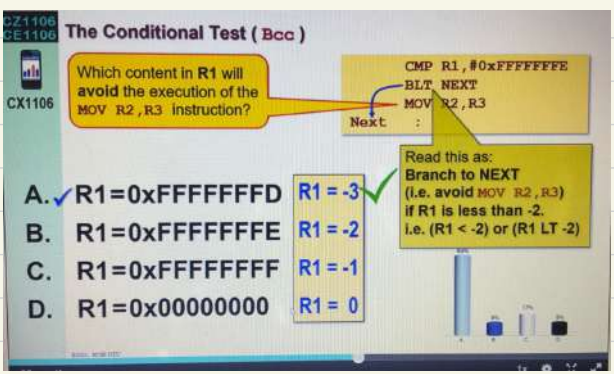
ADDS Loop Construct

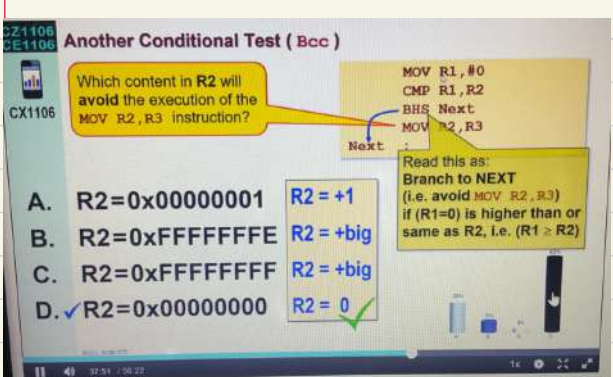


CMP Instruction



BCC Conditional Test





0 ≥ 1
0 ≥ big
0 ≥ big
0 ≥ 0 ✓

Review Question

Q1 – CALL

Qns: What registers may be affected by executing:

BL RegR1

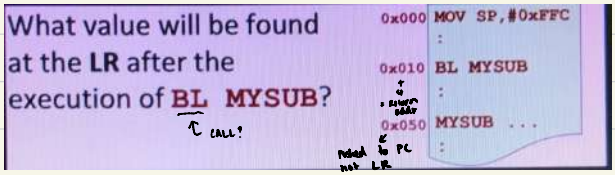
Ans: PC & LR

Take location of instruction "BL RegR1" + 4 store into Link Register (LR)

Then take RegR1 make a copy of the above memory location and put into PC

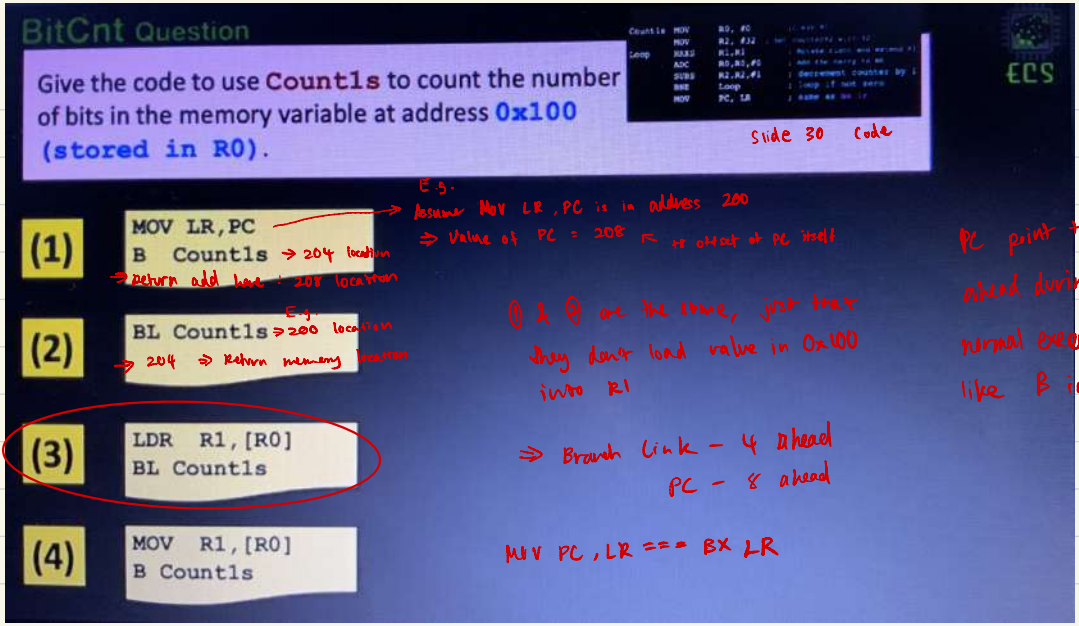
Stack Pointer not affected by BL instruction in this case

Q2 - CALL



Ans : 0x014

* Every instruction a fixed 4 bytes.



(1) "MOV LR, PC" places PC (address of MOV+8) to LR, followed by a branch

⇒ Actually the correct return address

However, value not loaded to R1 prior to subroutine

⇒ MOV LR,PC & Count1s = BL Count1s

(2) Have to prepare the parameter first before calling the Count1s subroutine

Count1s requires the parameter to be passed via the R1 register

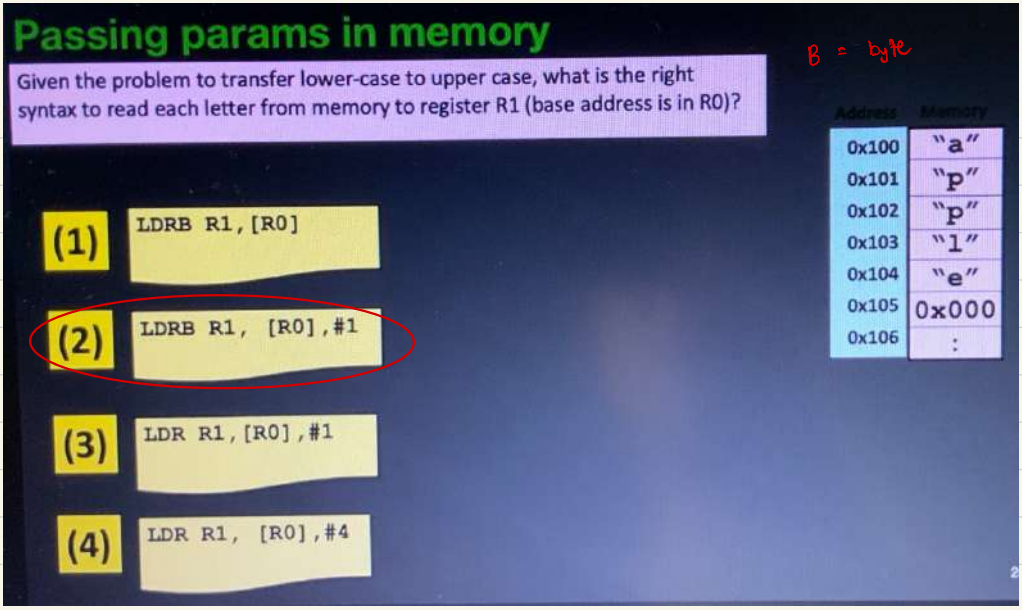
(3)

1. Set up parameter into R1 register
2. Call Count1s subroutine with BL instruction using the appropriate subroutine label

(3) MOV an invalid instruction; cannot be used to read from memory

Operands can be only

* (1) if another instruction was placed between MOV & branch, it would loop infinitely



(1) Byte is read but does not update R0 for the next instruction

⇒ increment R0 in another instruction

E.g. ADD R0, R0, #1

(2) instruction can read a byte, and increments R0 with 1 after fetching

⇒ by auto indexing

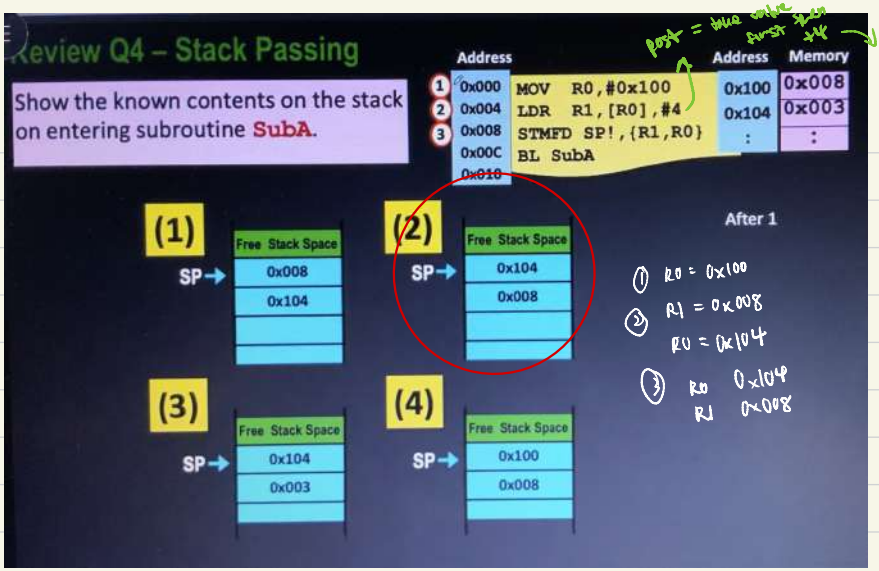
(3) instruction can read a 32-bit word, and increments R0 with 1 after fetching

"#1" will cause an error ⇒ runtime cause trying to get unaligned word

"0x100" → "0x101" ✗ "0x104" allowed

(4) instruction can read a 32-bit word, and increments R0 with 4 after fetching

"#4" will not cause an error



After 1

R0	0x100
R1	

After 2

R0	0x104
R1	0x008

LDR R1,[R0],#4

1) go to address in R0 (0x100), take the value (0x008) and put into R1
2) increment R0 by 4 (0x100 → 0x104)

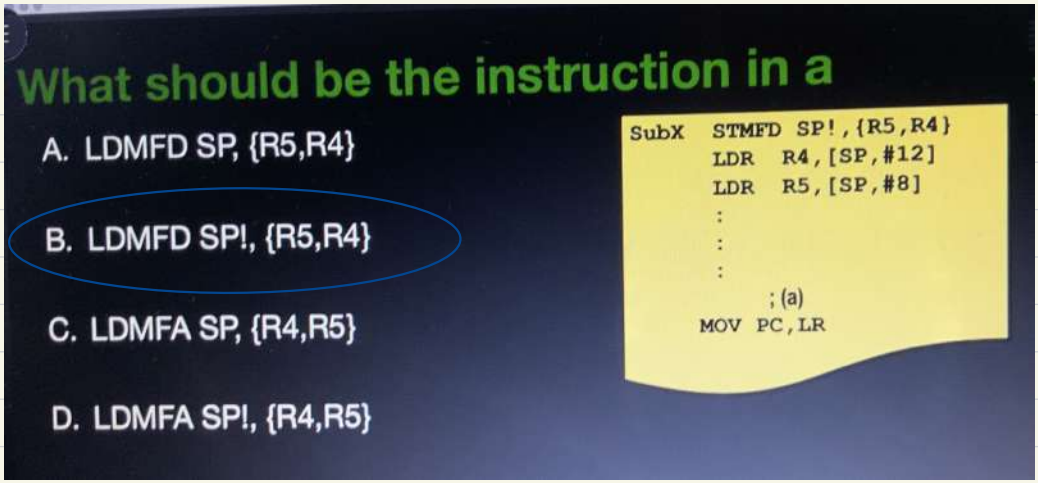
After 3

R0	0x104
R1	0x008

↓ ↓ put into stack

R0	0x104
R1	0x008

Clock cycles of STMFD
⇒ For 1 register, 3 cycles.
⇒ +1 cycle for each additional register



C & D : Since STMFD was used, C & D not valid here

A : does not update stack pointer

For the shown code segment, how are parameters A and B passed?

```
MOV   R0,#0x100
LDR   R1,[R0],#4
STMFD SP!,{R0,R1}
BL    Sub
```

Memory: 0x100 → A, 0x104 → B (addresses / values)

A. A reference, B value

B. A value, B value

C. A reference, B reference

D. A value, B reference

① R0 = 0x100

② R1 = A — value
R0 = 0x104 — reference

0x104 : reference of B

R1 has the value of A

0x100	A
0x104	← B

① R0 has address of A

② post index

R0	0x104
R1	A

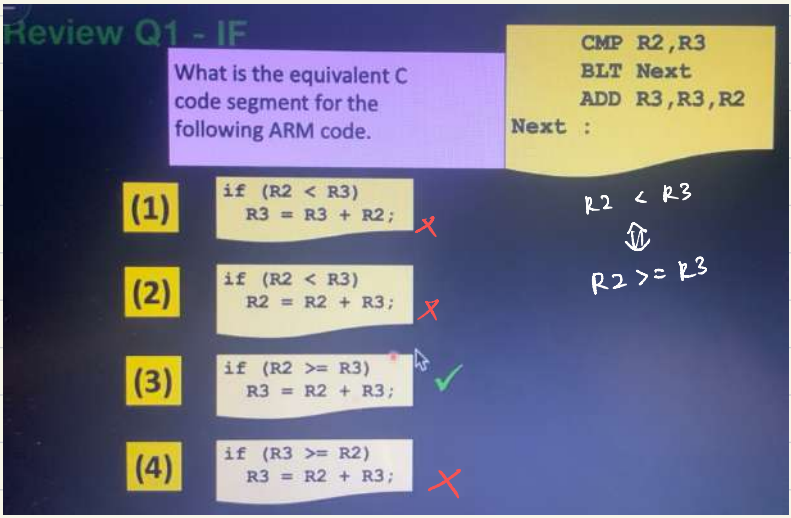
← value of 0x100 updated by 4

MOV doesn't mean always pass by ref, it is case paired with LDR, then one value one reference

The segment loads the content of memory address 0x100 to R1 → value of A

we store in the stack then the value of A and the value 0x104 (address of B)

19/2 lecture



(1) Need to use reverse condition for "<", which is ">=" or BGE for efficient implementation

(2) Same as (1) but in ADD, R3 is destination register. So ADD R3, R3, R2 gives R3 = R2 + R3

(3) reverse condition correct

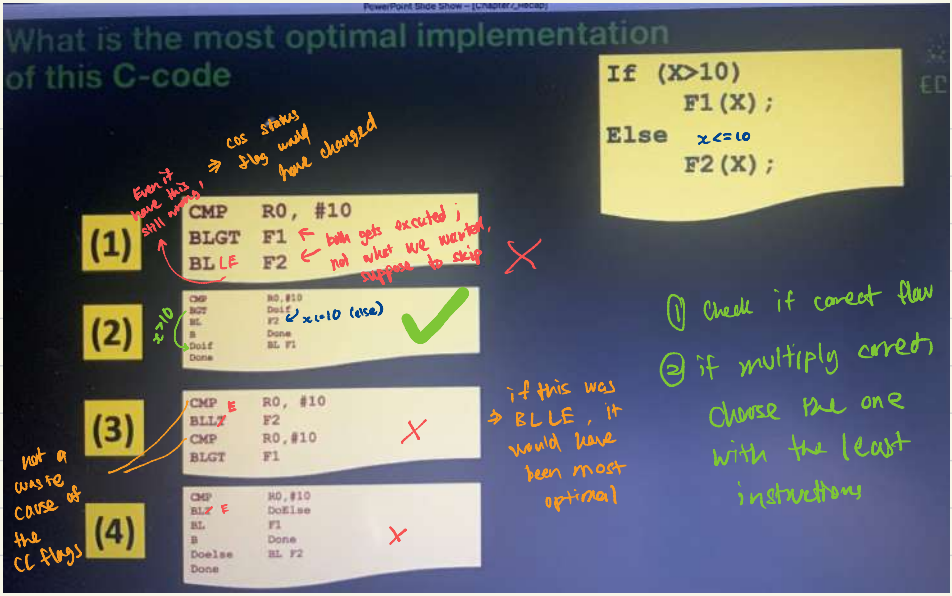
CMP R2, R3

↑

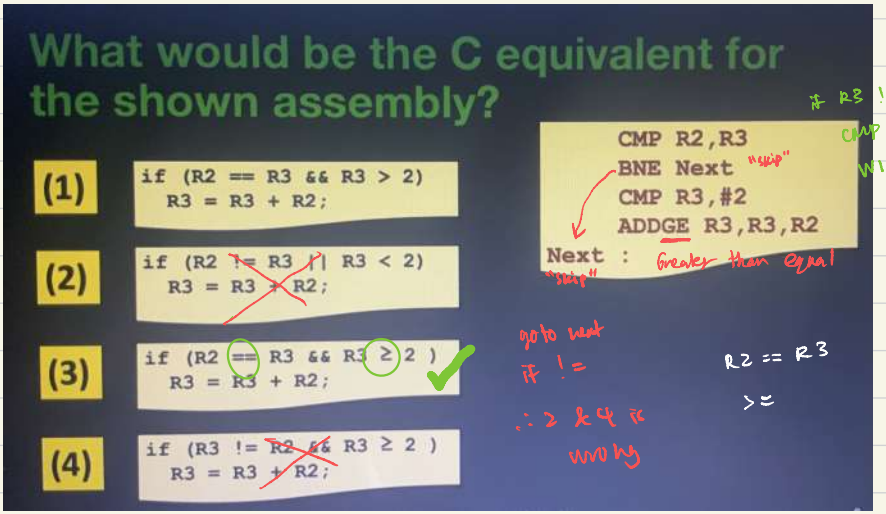
comes first

(4) reverse condition correct

but wrong order of R2, R3 in CMP instruction


PowerPoint Slide Show - [Chapter7_Recap]
What is the most optimal implementation of this C-code
If (X>10)
F1(X);
Else x<=10
F2(X);
(1)
CMP R0, #10
BLGT F1
BL LE F2
Even if have this still running,
cos status flag would have changed
both gets executed, not what we wanted, suppose to skip
(2)
CMP R0,#10
BGT Dolf
BL F2
B Done
Dolf BL F1
Done
x<=10 (else)
x>10
(3)
CMP R0, #10
BLLT F2
CMP R0,#10
BLGT F1
not a waste cause of the CC flags
(4)
CMP R0,#10
BLE DoElse
BL F1
B Done
Doelse BL F2
Done
if this was BL LE, it would have been most optimal
1 Check if correct flow
2 if multiply correct, choose the one with the least instructions

22/2 lecture

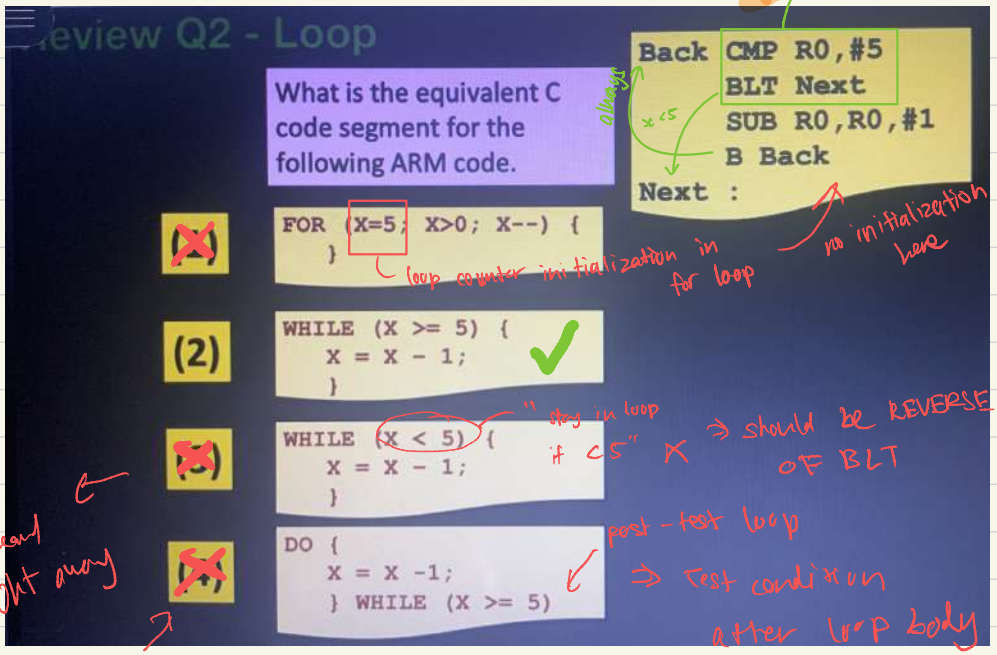


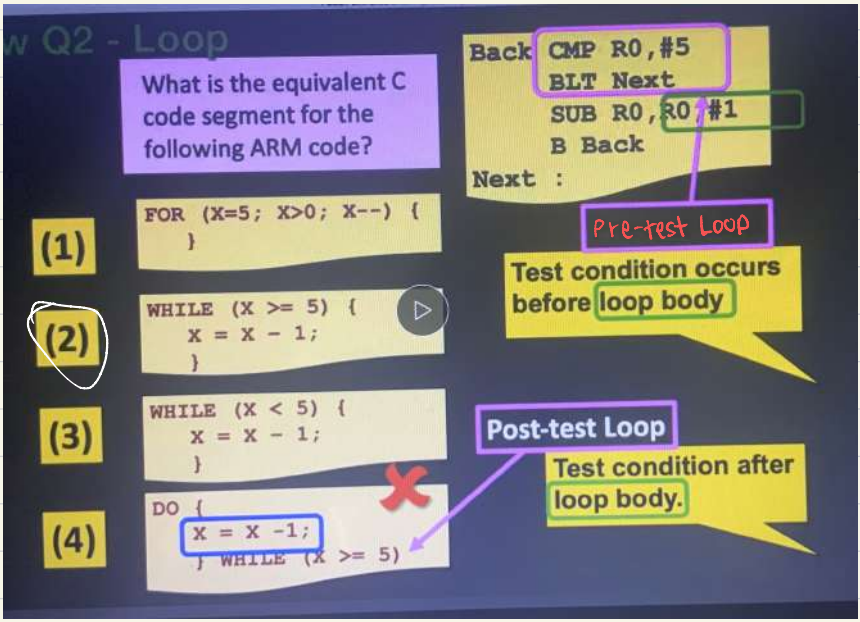
if && : negate first condition

if || : negate last condition

22/2 Lecture

Pretest is used in both loop constructs — stay looping if (x >= 5) [x = x-1]





What is the instruction to create a stack frame for 4 local variables (32-bit each)

A. ADD SP, SP, #16

B. SUB SP, SP, #16

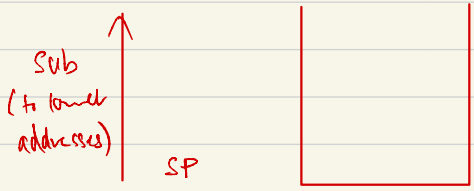
C. SUB SP, SP,#32

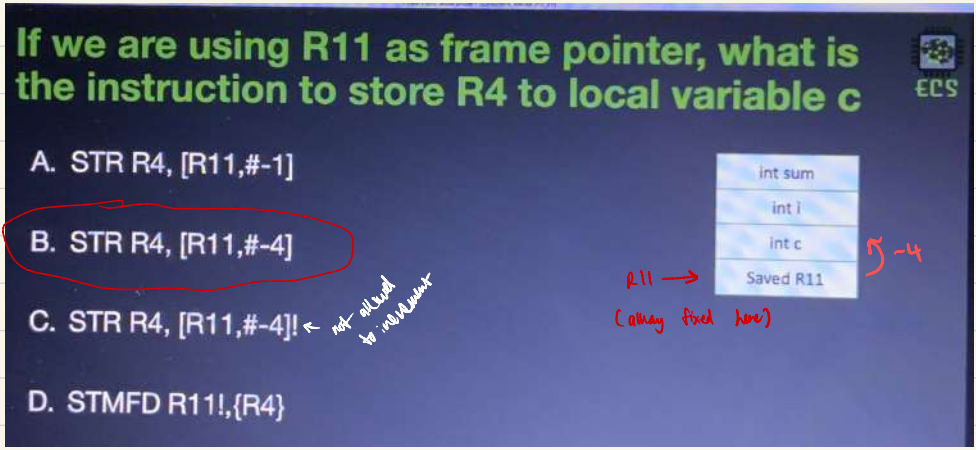
D. ADD SP,SP,#32

STR R11, [SP,#-4]!
MOV R11, SP

each is 4 bytes

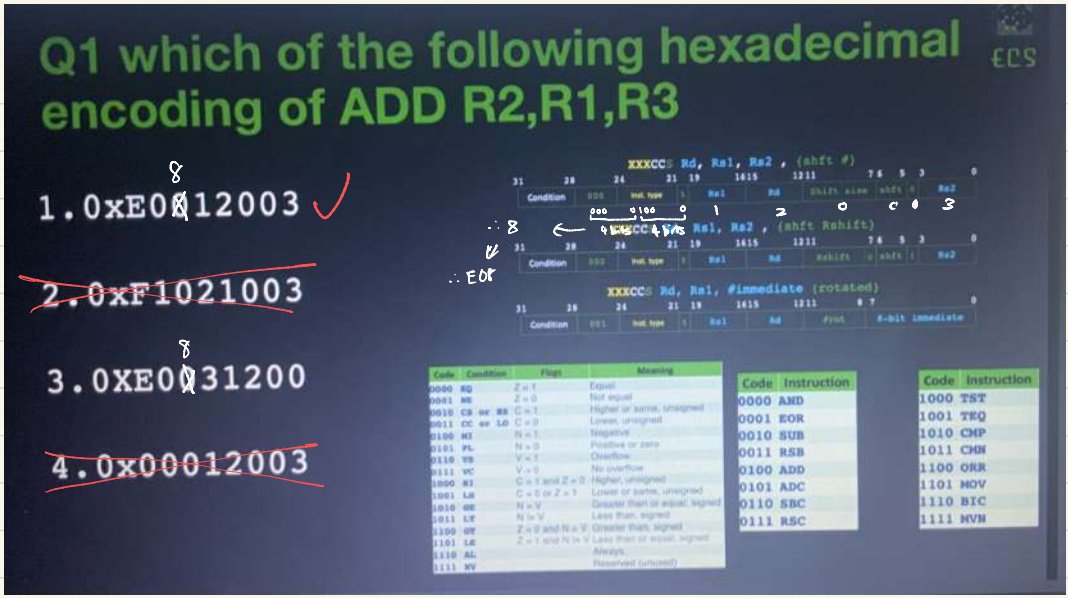
∴ 4 + 4 = 16 bytes





if r11 was a stack pointer, then we would have to update it

26/2 Lecture



Addresses of registers:

R0 – 0000

R1 – 0001

R2 – 0010

R3 – 0011

⋮

ADD – no condition

⇒ does not mean condition = 0000

↳ cause 0000 indicates EQ

⇒ So must be condition = 1110

↳ aka (E), indicates "Always"

XXXCCS Rd, Rs1, Rs2 , {shft #}

31–28	27–24	23–21	20	19–16	15–12	11–7	6–5	4	3–0
Condition	000	Inst. type	S	Rs1	Rd	Shift size	shft	0	Rs2
1110	000	0100	0	1	2	0	0	0	3

1110 ↑ aka E, "always"

0100 ↑ ADD code

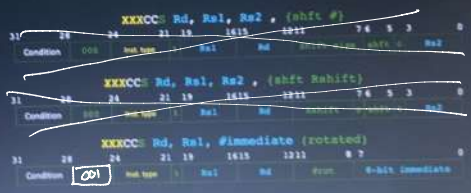
0 ↑ carry is ADD, not ADDS

0 0 0 ↑ no shifts

Q2 which of the following hexadecimal encoding of RSBEQS R2,R1,#5

← 8 bit immediate value

1. 0xF1012005
2. 0x02721005 ✓
3. 0X01321005
4. 0x01212005



Code	Instruction
0000	AND
0001	EOR
0010	SUB
0011	RSB
0100	ADD
0101	ADC
0110	SBC
0111	RSC

Code	Instruction
1000	TST
1001	TEQ
1010	CMP
1011	CMN
1100	ORR
1101	MOV
1110	BIC
1111	MVN

0000 001 0011 1 1 2 5

XXXCCS Rd, Rs1, #immediate (rotated)

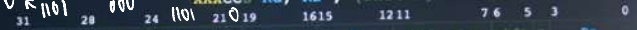
31–28	27–24	23–21	20	19–16	15–12	11–8	7–0
Condition	001	Inst. type	S	Rs1	Rd	#rot	8-bit immediate

000 if 3rd operand is a register

Q3 which of the following hexadecimal encoding of MOVLE R7,R4

1. 0xD0D74000
2. 0xD0D70004
3. 0XD1A70004 ✓
4. 0xD1A74000

XXXCCS Rd, Rs , {shft #}



XXXCCS Rd, Rs , {shft #}

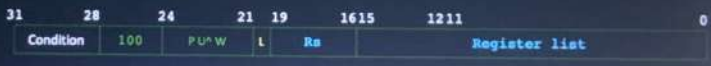
XXXCC Rd, #immediate (rotated)

Code	Condition	Flags	Meaning
0000	EQ	Z = 1	Equal
0001	NE	Z = 0	Not equal
0010	CS or HS	C = 1	Higher or same, unsigned
0011	CC or LO	C = 0	Lower, unsigned
0100	MI	N = 1	Negative
0101	PL	N = 0	Positive or zero
0110	VS	V = 1	Overflow
0111	VC	V = 0	No overflow
1000	HI	C = 1 and Z = 0	Higher, unsigned
1001	LS	C = 0 or Z = 1	Lower or same, unsigned
1010	GE	N = V	Greater than or equal, signed
1011	LT	N != V	Less than, signed
1100	GT	Z = 0 and N = V	Greater than, signed
1101	LE	Z = 1 and N != V	Less than or equal, signed
1110	AL		Always
1111	NV		Reserved (unused)

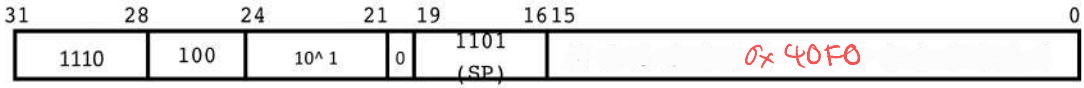
Code	Instruction
0000	AND
0001	EOR
0010	SUB
0011	RSB
0100	ADD
0101	ADC
0110	SBC
0111	RSC

Code	Instruction
1000	TST
1001	TEQ
1010	CMP
1011	CMN
1100	ORR
1101	MOV
1110	BIC
1111	MVN

Q4 If the registers list field is encoded with 0X40F0, what register list this represents?



1. {R4,R15}
2. {R4-R7,LR}
3. {R1,R8-R11}
4. Don't know!



PC LR SP R12 R11 R10 R9 R8 R7 R6 R5 R4 R3 R2 R1 R0

0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0

R4 - R7 , LR

Mock Quiz

(1)

0xF3
0x72
0x82

most negative short int

⇒ short int occupies 2 bytes (2 memory locations)

⇒ Ans: 8272

Little endian

: 8272F3

(2)

0x100	0x FF
	0x 00
	0x 00
	0x FF

LDR R0, [R0]

ORR R2, R1, R0

Initial: R0 = 0x100

R1 = 0xAABBCCDD

Use calculator OR

⇒ 32 bit, Unsigned

⇒ Ans: FF BBCCFF

(3)

```
Start MOV  R1,#-5
      SUB  R0,R0,R0
Loop  CMP  R1,#1
      BGT  Exit
      ADD  R0,R0,R1
      ADD  R1,R1,#2
      B    Loop
Exit  ADD  R0,R0,R1
Done  END
```

greater than

What is R0 at the end?

Start: R1 = −5 (not gt, go loop) (−5 > 1)

R0 = 0

Loop 1: R0 = −5 (not gt, go loop) (−3 > 1)

R1 = −3

Loop 2: R0 = −8 (not gt, go loop) (−1 > 1)

R1 = −1

Loop 3: R0 = −9 (not gt, go loop) (1 > 1)

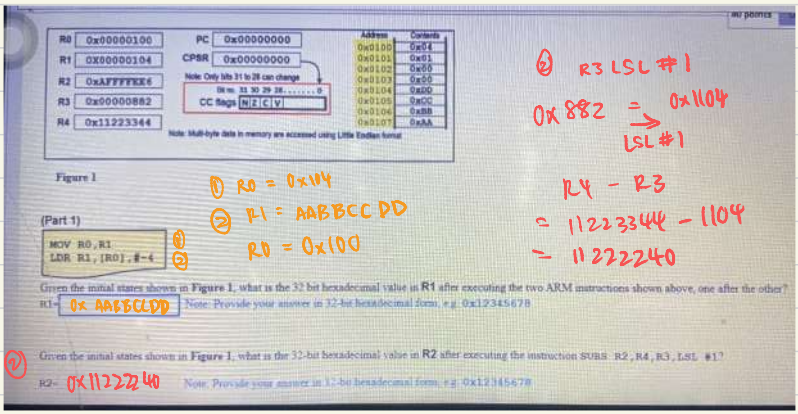
R1 = 1

Loop 4: R0 = −8 (gt, go loop) (3 > 1)

R1 = 3

Exit: R0 = −5

2

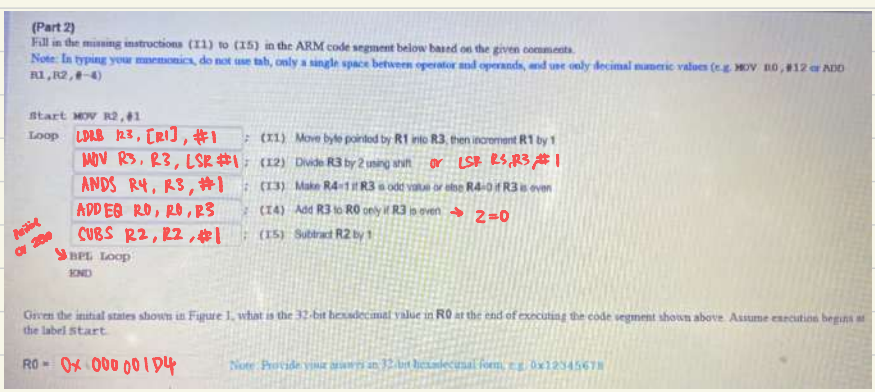


① R0 = 0x104
② R1 = AABBCCDD
R0 = 0x100

② R3 LSL #1
0x882 → 0x1104
LSL #1

R4 - R3
= 11223344 - 1104
= 11222240

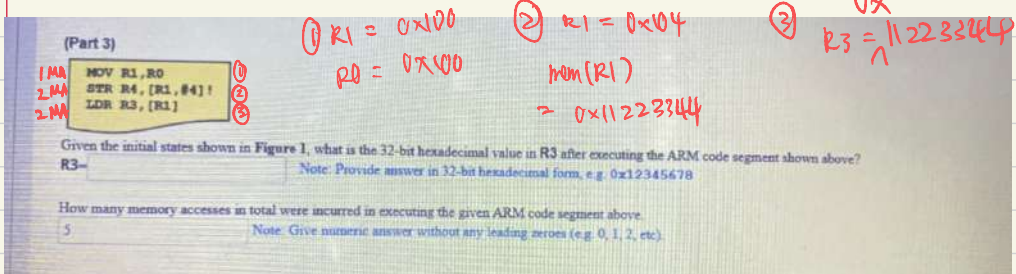
can use ctrl



Initial or zero

or LSR R3,R3,#1

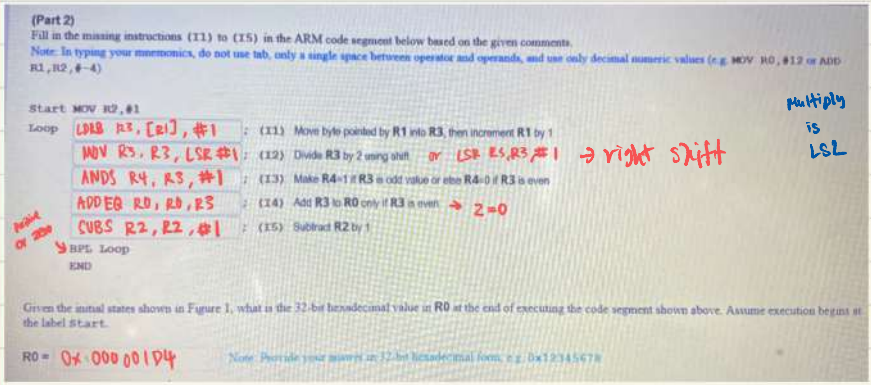
2=0



① R1 = 0x100
R0 = 0x100

② R1 = 0x104
mem(R1)
= 0x11223344

③ R3 = 0x11223344



(Part 2)

Fill in the missing instructions (I1) to (I5) in the ARM code segment below based on the given comments.

Note: In typing your mnemonics, do not use tab, only a single space between operator and operands, and use only decimal numeric values (e.g. MOV R0,#12 or ADD R1,R2,#-4)

```
Start MOV R2,#1
Loop  LDRB R3,[R1],#1   ; (I1) Move byte pointed by R1 into R3, then increment R1 by 1
      MOV R3,R3,LSR#1   ; (I2) Divide R3 by 2 using shift
      ANDS R4,R3,#1     ; (I3) Make R4=1 if R3 is odd value or else R4=0 if R3 is even
      ADDEQ R0,R0,R3    ; (I4) Add R3 to R0 only if R3 is even
      SUBS R2,R2,#1     ; (I5) Subtract R2 by 1
      BPL Loop
      END
```

Given the initial states shown in Figure 1, what is the 32-bit hexadecimal value in R0 at the end of executing the code segment shown above. Assume execution begins at the label Start.

R0 = 0x000001D4

Note: Provide your answer in 32-bit hexadecimal form, e.g. 0x12345678

multiply is LSL

LSR R3,R3,#1 → right shift

Repeat or Loop

Initial: R1 = 0x104 R0 = 0x100

R2 = 0x1

R3 = 0x882

Loop 1: R1 = #0x105

R3 = DD (dec 221 odd))

R3 ÷ 2 = DD ÷ 2 = 0x6E (right shift)

Z=0 → 0x6E AND 1 = 0 = R4 → Z = 1

R0 = R0 + R3 = 0x16E

R2 = 0 (Loop break)

Loop 2: R1 = #0x106

R3 = CC

R3 ÷ 2 = CC ÷ 2 = 0x66

0x66 AND 1 = 0 = R4 → Z = 1

```
AND 1100110
    0000001
    -------
    0000000
```

R0 = R0 + R3 = 0x16E + 0x66

= 0x1D4

R2 = -1 (Exit)