
Mitsubishi M16C Instruction Set Architecture

Lecture 2

Today

Learn about Mitsubishi (Renesas) processor

- Lecture covers ISA, derived from **Assembler Language Programming Manual** (M16C_Assembler.pdf MALPM Ch. 1, 2)
 - Registers: General Purpose, Control
 - Instruction Set
 - Addressing Modes
 - Memory Map

Reading for Next Week

Software Manual (M16C_Software_Manual.pdf):

pp.1-32

- Use chapter 3 as a reference. *You are responsible for this material – not memorizing it, but instead being able to figure out what an instruction does, or finding an instruction to do something*

Data Sheet

(M16C62_Hardware_Manual_rev1.20.pdf)

- pp. 1-27

Simple Memory Organization

$k \times m$ array of stored bits (k is usually 2^n)

Address

- unique (n -bit) identifier of location

Contents

- m -bit value stored in location

Basic Operations:

LOAD

- read a value from a memory location

STORE

- write a value to a memory location

0000	
0001	
0010	
0011	00101101
0100	
0101	
0110	
	⋮
1101	10100010
1110	
1111	

Simple Memory Organization

Viewed as a large, single-dimensional array

A memory address is an index into the array

"Byte addressing" means that the index points to a byte of memory

0	8 bits of data
1	8 bits of data
2	8 bits of data
3	8 bits of data
4	8 bits of data
5	8 bits of data
6	8 bits of data

...

Memory Organization

Bytes are nice, but most data items use larger "words"

- For M30626, a word is 16 bits or 2 bytes.

0	16 bits of data
2	16 bits of data
4	16 bits of data
6	16 bits of data

(registers also hold 16 bits of data)

2^{16} bytes with byte addresses from 0, 1, 2 to $2^{16}-1$

2^{15} words with byte addresses 0, 2, 4, ... $2^{16}-2$

...

Endianness

Big endian: most significant byte is stored at the lowest byte address

◆ Ex: 68000, PowerPC, Sun SPARC

0	12
1	34
2	56
3	78
4	AB
5	CD
6	EF
7	01

BOTH store:
word 12345678 is at location 0,
word ABCDEF01 is at location 4

○ **Little endian: least significant byte is stored at the lowest address**

◆ Ex: x86, DEC VAX, Alpha

0	78
1	56
2	34
3	12
4	01
5	EF
6	CD
7	AB

- Most of the time we will avoid this issue in class by only loading/storing words or loading/storing bytes
- If two processors with different conventions use a local area network, a disk drive, etc., YOU need to pay attention to endianness

Big Endian-What does it look like?

Imagine you have the following hexadecimal values:

- 0x11 22 33 44
- 0x55 66 77 88
- 0xAA BB CC DD
- 0xEE FF 00 99
- 0x01 23 45 67

And we put them in memory, starting at memory address 0x10000000.

What would it look like?

0x1000 0000	11 22 33 44	Shown as “Big Endian”
0x1000 0004	55 66 77 88	
0x1000 0008	AA BB CC DD	
0x1000 000C	EE FF 00 99	
0x1000 0010	01 23 45 67	
0x1000 0014		

Little Endian -What does it look like?

Imagine you have the following hexadecimal values:

- 0x11 22 33 44
- 0x55 66 77 88
- 0xAA BB CC DD
- 0xEE FF 00 99
- 0x01 23 45 67

And we put them in memory, starting at memory address 0x10000000.

What would it look like?

0x1000 0000	44 33 22 11	Shown as “Little Endian”
0x1000 0004	88 77 66 55	
0x1000 0008	DD CC BB AA	
0x1000 000C	99 00 FF EE	
0x1000 0010	67 45 23 01	
0x1000 0014		

Big Endian vs. Little Endian

0x1000	0000	11 22 33 44	Shown as “Big Endian”
0x1000	0004	55 66 77 88	
0x1000	0008	AA BB CC DD	
0x1000	000C	EE FF 00 99	
0x1000	0010	01 23 45 67	

0x1000	0000	44 33 22 11	Shown as “Little Endian”
0x1000	0004	88 77 66 55	
0x1000	0008	DD CC BB AA	
0x1000	000C	99 00 FF EE	
0x1000	0010	67 45 23 01	

Data Formats for the M30626

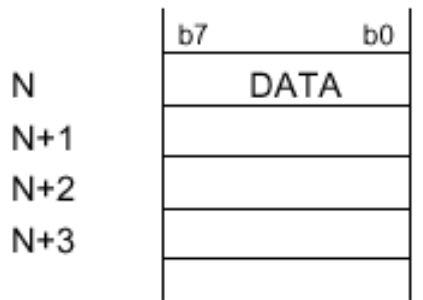
Byte

- 8 bits
- signed & unsigned
- .B suffix for instruction

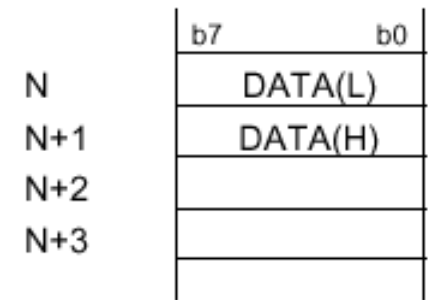
Is the M30626 big or little endian?

Word

- 16 bits
- signed & unsigned
- .W suffix



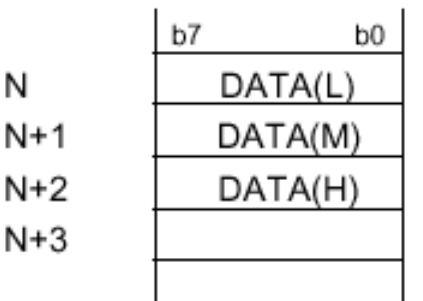
Byte (8-bit) data



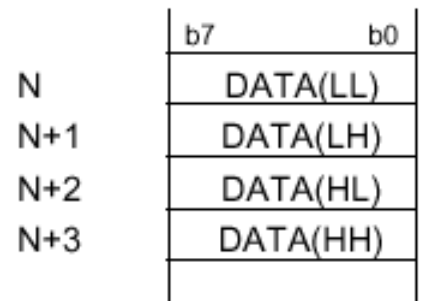
Word (16-bit) data

Address & longword

- Limited to specific instructions



20-bit (Address) data

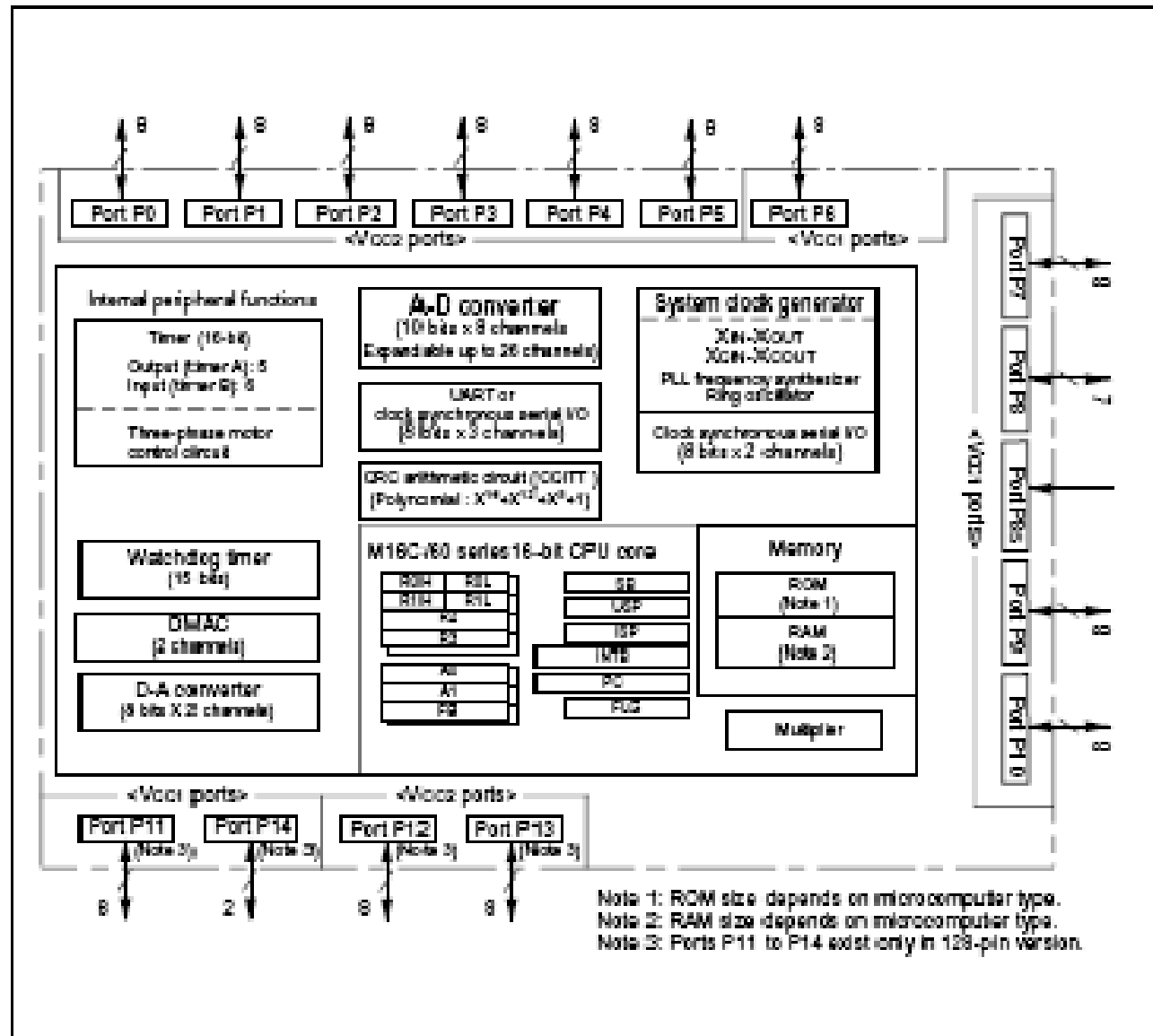


Long Word (32-bit) data

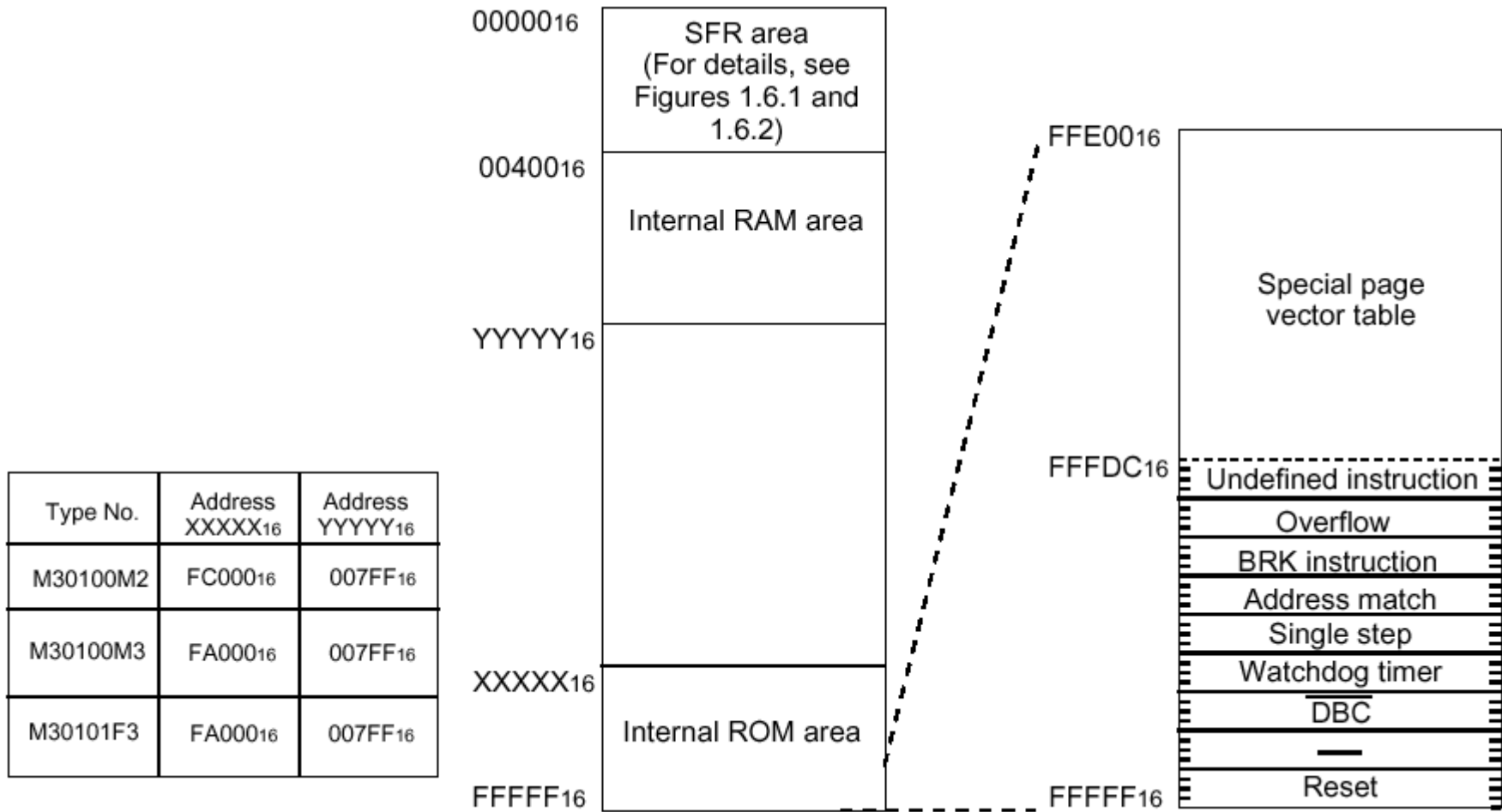
Review of the M30626 Architecture

Microcontroller has:

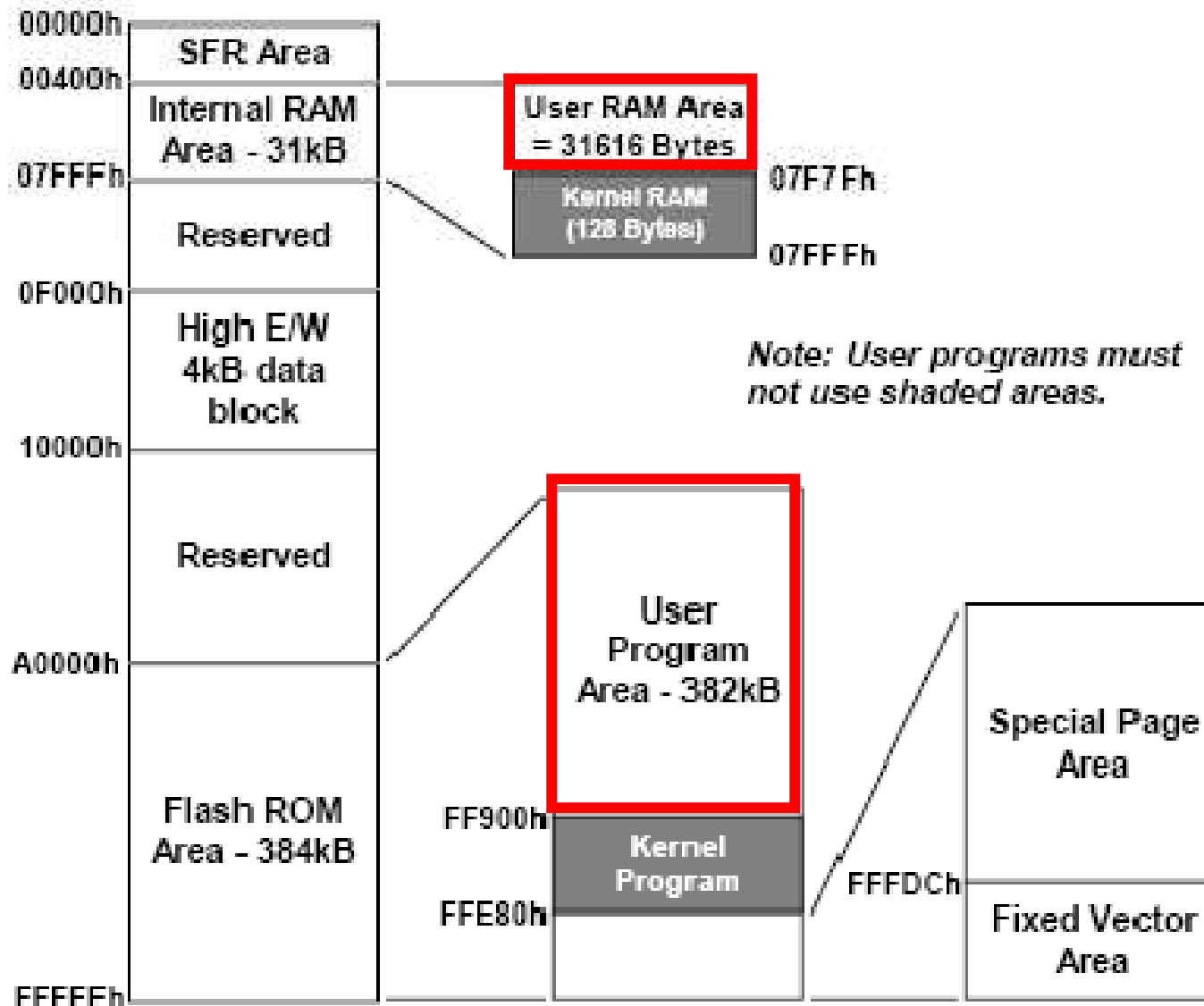
- General Purpose Registers
- RAM
- Flash
- EEPROM
- Digital Ports
- Analog Ports
- Timers
- Oscillator
- DMA Controller
- Reliability and safety



General Memory Map



Memory Map for QSK62P Plus microcontroller



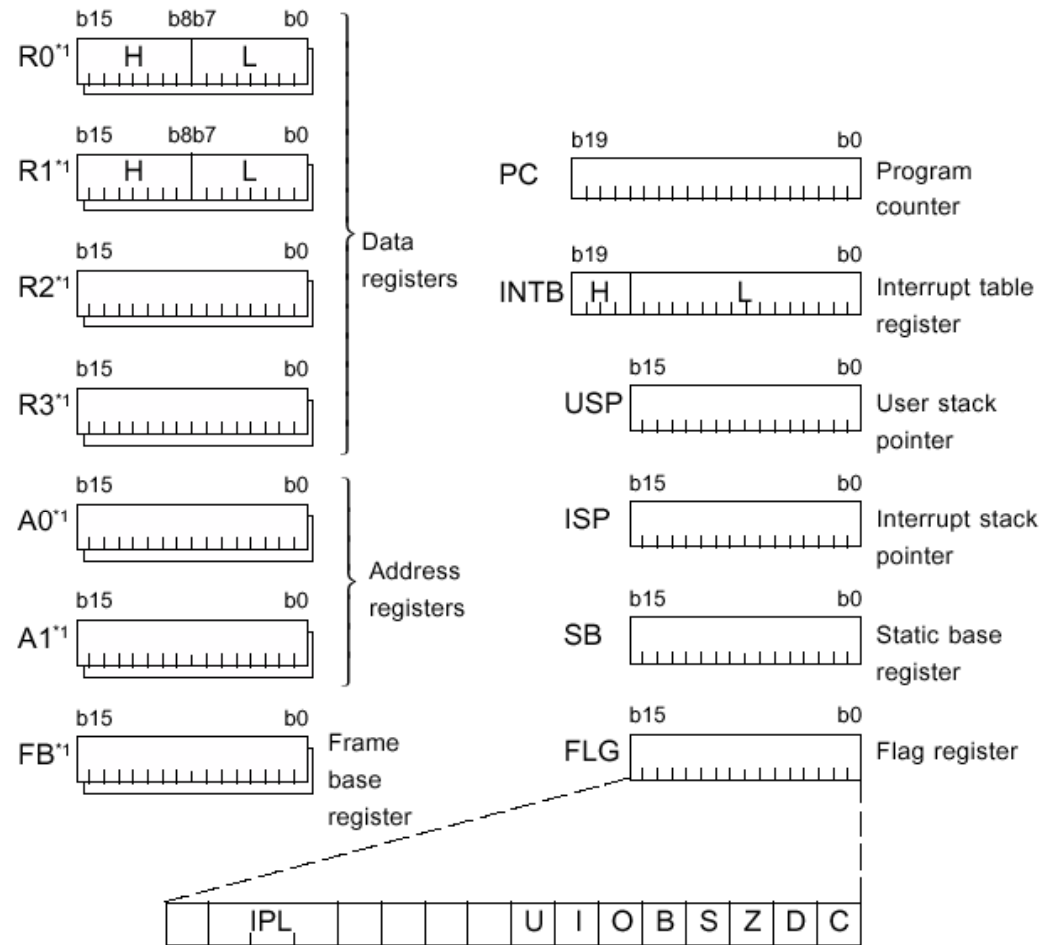
M16C Registers

4 16-bit data registers R0-R3

- Can also access high and low bytes of R0 and R1: R0H, R0L
- Can also access pairs of registers as 32-bit registers: R2R0, R3R1

2 16-bit address registers A0 & A1

- Can also access pair of registers as 32-bit register: A1A0



*1 These registers have two register banks.

Special Registers

SP: Stack Pointer – for accessing call stack

- USP: User code
- ISP: Interrupt code

FB: Frame Base – for accessing frame on call stack

SB: Static Base

INTB: Interrupt table pointer

Addressing Modes

See Ch. 2 of Software Manual for details

Immediate – provide the 8, 16 or 20 bit value

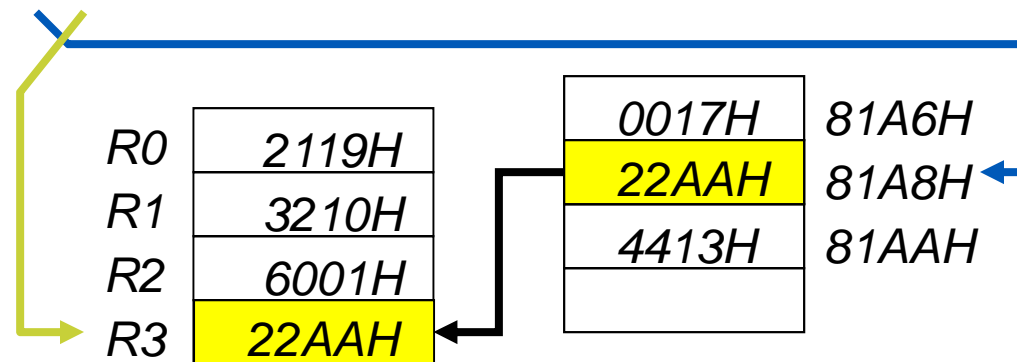
Register Direct – provide the name of the register

– MOV.B #-29, R0H

Absolute – provide the address of the operand

– MOV.W R3, 213AH

– MOV.W 81A8H, R3

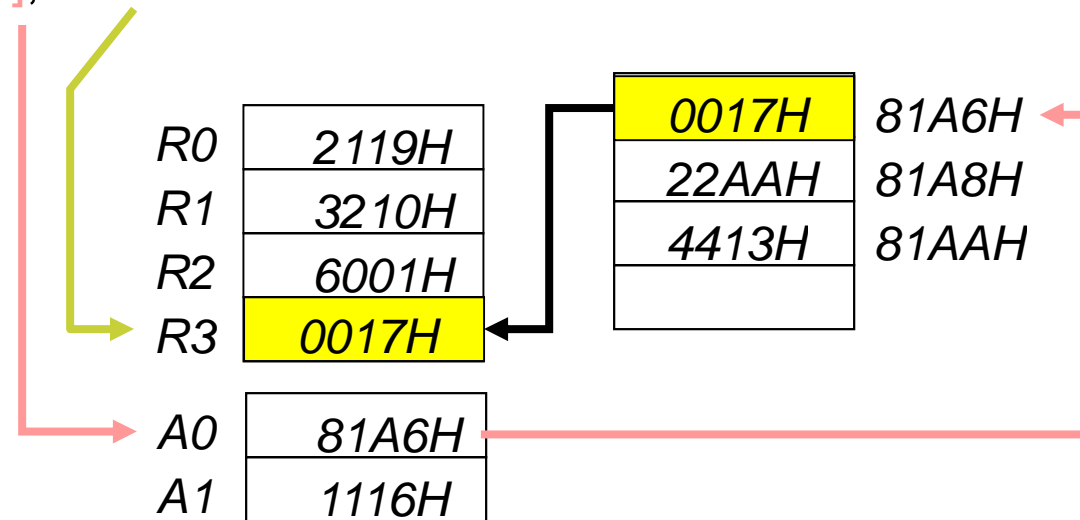


Address Register Indirect

A

Address Register Indirect – provide the name of the address register which points to the operand

– MOV.W [A0], R3



Constraints

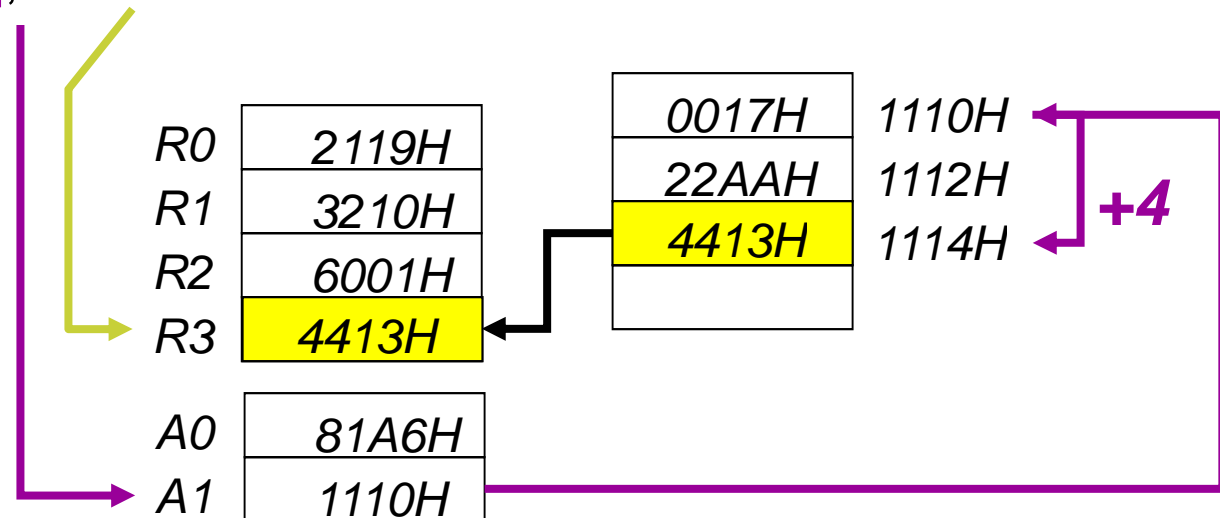
– Can use address registers A0 and A1

Address Register Relative

A

Address Register Relative – as with ARI, but also provide a displacement (offset) from the address register

– MOV.W 4[A1], R3



Constraints

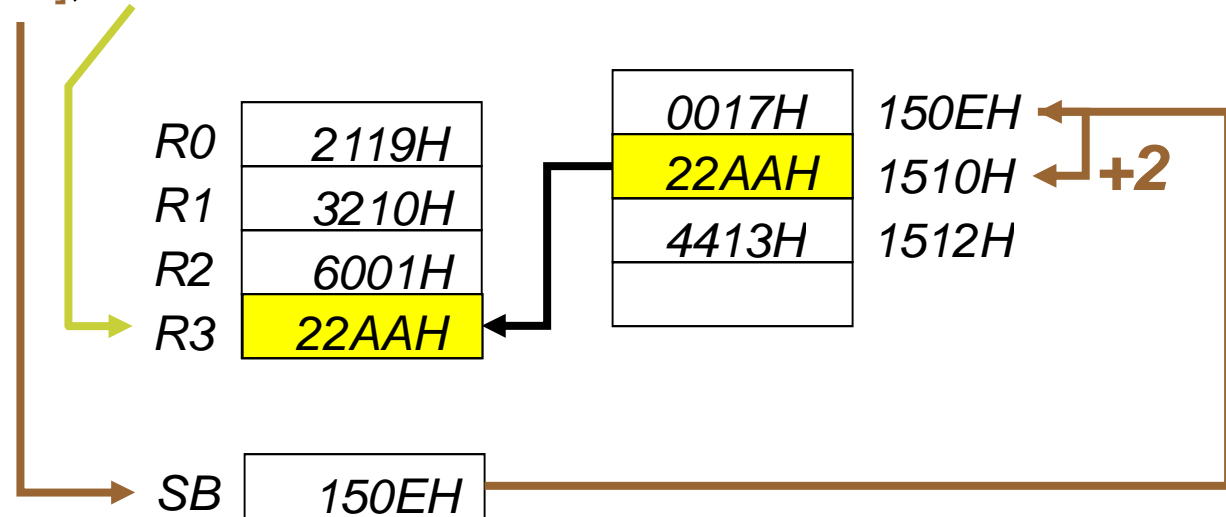
- Can use address registers A0 or A1
- Displacement can range from 0 to FFFFH

Static Base Pointer Relative

A

Static Base Pointer Relative – as with ARR, but use the SB as the base

– MOV.W 2[SB], R3



Constraints

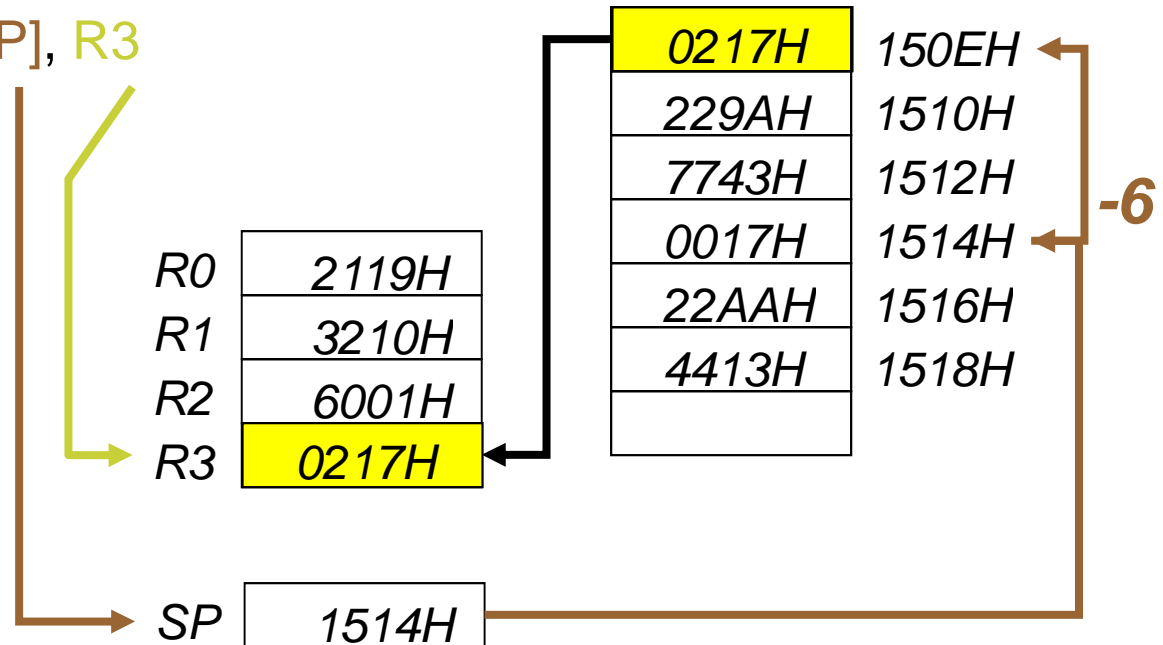
- Can only use SB
- Displacement can range from 0 to FFFFH

Frame Base/Stack Pointer Relative

A

Frame Base/Stack Pointer Relative – as with ARR, but use the FB or SP register as the base

– MOV.W -6[SP], R3



Constraints

- Can only use FB and SP
- Signed displacement can range from 80H to 7FH (-128 to +127)

Instruction Classes

Data Transfer

Arithmetic and Logic

Control Transfer

Other

Data Transfer Instructions

Not load-store architecture

- Transfer instructions
- Push/pop instructions
- Extended data area transfer instructions
- 4-bit transfer instructions
- Exchange between register and register/
memory instruction
- Conditional transfer instructions

MOV, MOVA
PUSH, PUSHM, PUSHA / POP, POPM
LDE, STE
MOVDir
XCHG

STZ, STNZ, STZX

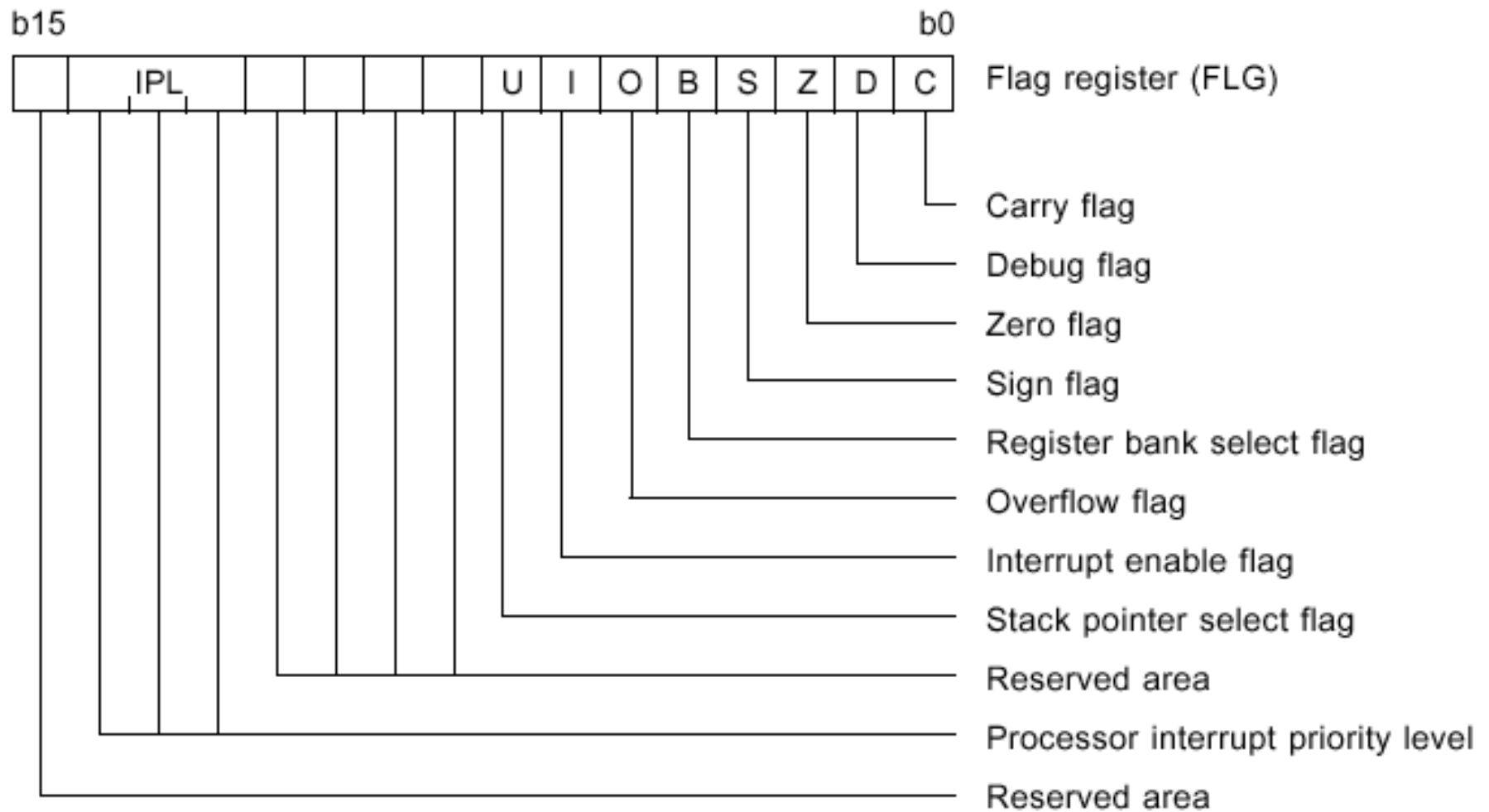
Arithmetic and Logic Instructions

• Add instructions	ADD, ADC, ADCF
• Subtract instructions	SUB, SBB
• Multiply instructions	MUL, MULU
• Divide instructions	DIV, DIVU, DIVX
• Decimal add instructions	DADD, DADC
• Decimal subtract instructions	DSUB, DSBB
• Increment/decrement instructions	INC / DEC
• Sum of products instruction	RMPA
• Compare instruction	CMP
• Others (absolute value, 2's complement, sign extension)	ABS, NEG, EXTS
• Logic instructions	AND, OR, XOR, NOT
• Test instruction	TST
• Shift/rotate instructions	SHL, SHA / ROT, RORC, ROLC

Control Transfer

- | | |
|--|--------------|
| • Unconditional branch instruction | JMP |
| • Conditional branch instruction | JCnd |
| • Indirect jump instruction | JMPI |
| • Special page branch instruction | JMPS |
| • Subroutine call instruction | JSR |
| • Indirect subroutine call instruction | JSRI |
| • Special page subroutine call instruction | JSRS |
| • Subroutine return instruction | RTS |
| • Add (subtract) and conditional branch instructions | ADJNZ, SBJNZ |

Flag Register and Conditions



Conditional Jumps

Mnemonic	Description Format	Explanation
JCnd	JCnd label	Jumps to label if condition is true or executes next instruction if condition is false.

Cnd	True/false determining conditions (14 conditions)	
GEU/C	$C = 1$	Equal or greater/ Carry flag = 1
GTU	$C = 1 \ \& \ Z = 0$	Unsigned and greater
EQ/Z	$Z = 1$	Equal/ Zero flag = 1
N	$S = 1$	Negative
LE	$(Z = 1) \ \ (S = 1 \ \& \ O = 0) \ \ (S = 0 \ \& \ O = 1)$	Equal or signed and smaller
O	$O = 1$	Overflow flag = 1
GE	$(S = 1 \ \& \ O = 1) \ \ (S = 0 \ \& \ O = 0)$	Equal or signed and greater
LTU/NC	$C = 0$	Smaller/ Carry flag = 0
LEU	$C = 0 \ \ Z = 1$	Equal or smaller
NE/NZ	$Z = 0$	Not equal/ Zero flag = 0
PZ	$S = 0$	Positive or zero
GT	$(S = 1 \ \& \ O = 1 \ \& \ Z = 0) \ \ (S = 0 \ \& \ O = 0 \ \& \ Z = 0)$	Signed and greater
NO	$O = 0$	Overflow flag = 0
LT	$(S = 1 \ \& \ O = 0) \ \ (S = 0 \ \& \ O = 1)$	Signed and smaller

Assembler Language Programming Manual, Sect. 2.6.1 and jump instruction definitions (p. 80)

Example:

```
CMP.W R1, R2
; set cond. flags
; based on R2-R1
JGTU Label2
; Jump if R1>R2
```

Range of jump : -127 to +128 (PC relative) for GEU/C, GTU, EQ/Z, N, LTU/NC, LEU, NE/NZ, and PZ
 -126 to +129 (PC relative) for LE, O, GE, GT, NO, and LT

Other Instructions

- Control register manipulate instructions
- Flag register manipulate instructions
- OS support instructions
- High-level language support instructions
- Debugger support instruction
- Interrupt-related instructions
- External interrupt wait instruction
- No-operation instruction

LDC, STC, LDINTB, LDIPL, PUSHC, POPC
FSET, FCLR
LDCTX, STCTX
ENTER, EXITD
BRK
REIT, INT, INTO, UND
WAIT
NOP