CHAPTER 5

X86 INSTRUCTION SET

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

fit@hcmus

fit@hcmus

REMIND

- CISC
- MIPS-32 bits operations

PREREQUITES

- ☐ Take a view on tutorial video
- ☐ Install NASM already

What will you learn?

- ☐ Inside a CPU Intel 8080/8086
- ☐ Registers
- ☐ Data addressing modes
- ☐ Operations
- ☐ Interrupt (I/O calling)
- ☐ Instruction Encoding
- ☐ Procedure

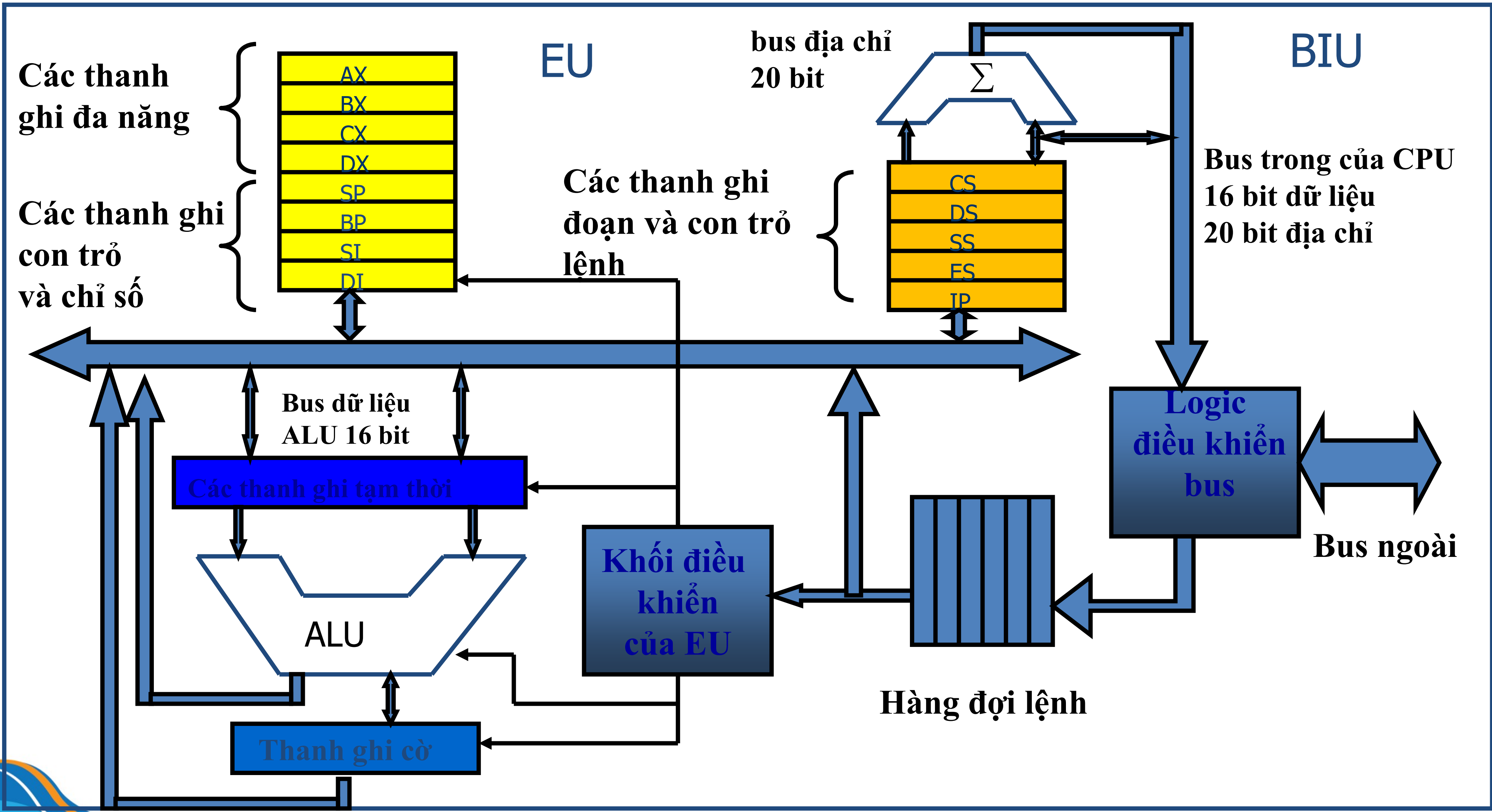
X86 Architecture

- Complexity
  - instructions from 1 to 17 bytes long
  - one operand *must* act as both a source and destination
  - one operand *may* come from memory
  - several complex addressing modes
- Saving grace:
  - the most frequently used instructions are not too difficult to build
  - compilers avoid the portions of the architecture that are slow

The Intel x86 ISA

- 1978: The Intel 8086 is announced (16 bit architecture)
- 1980: The 8087 floating point coprocessor is added
- 1982: The 80286 increases address space to 24 bits, +instructions
- 1985: The 80386 extends to 32 bits, new addressing modes
- 1989-1995: The 80486, Pentium, Pentium Pro add a few instructions (mostly designed for higher performance)
- 1997: MMX is added
- I486: pipelined, on chip cache
- 2017: Core i9 (x86-64), instruction set extensions SSE4.1, SSE4.2, AVX2

Inside a 8086-CPU



Register Files

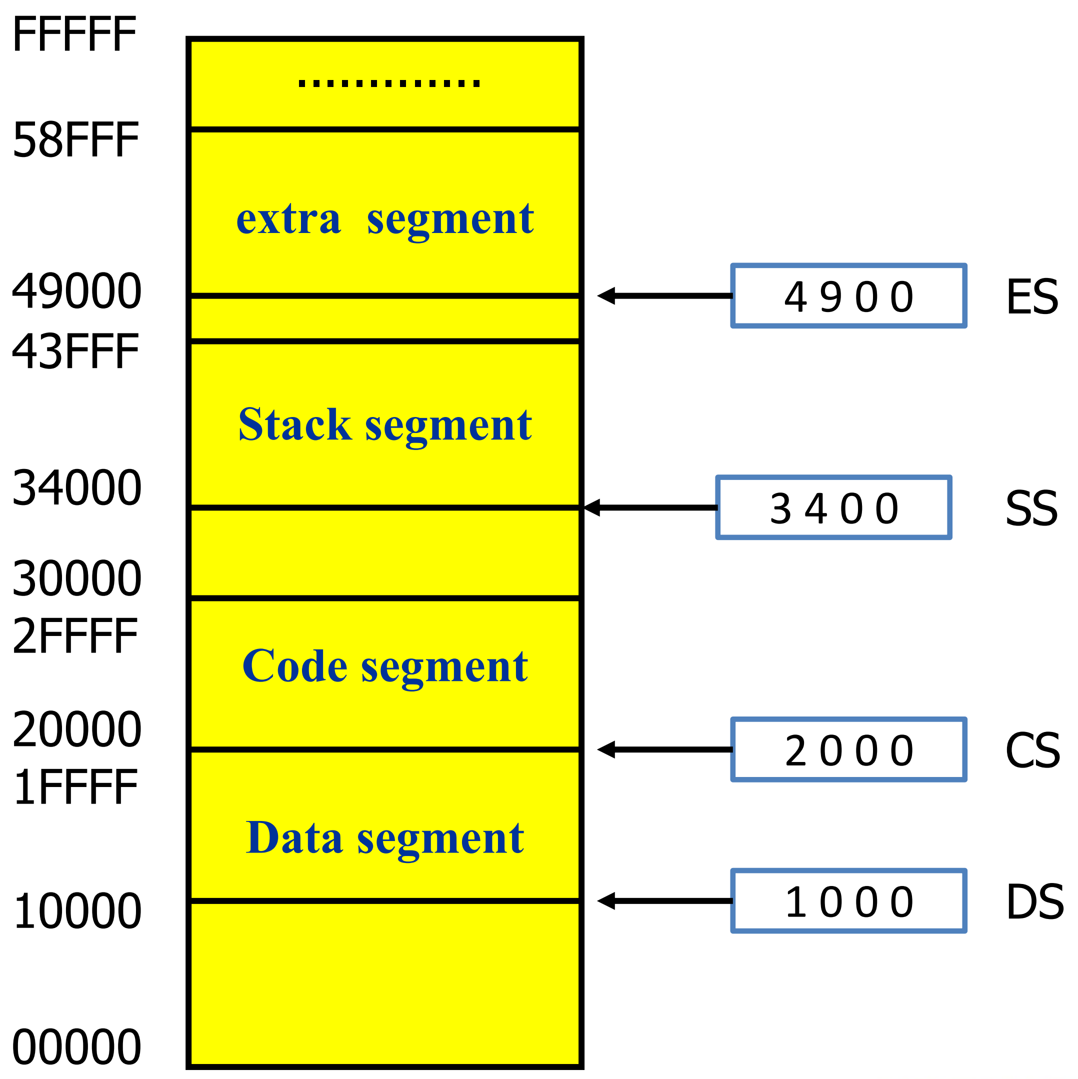
- *H register holds the 8-bit high,*L holds the 8-bit low of *X register
- AX (accumulator): hold the result of operations
- BX (base): base address
- CX (count): the number of times (Loop). CL saves the number of bits in shift / rotate bit operations
- DX (data): combine with AX to hold the result of multiplication/ division. DX also holds the address of in/output port (IN/OUT)

		8-bit high	8-bit low
General purpose registers	AX	AH	AL
	BX	BH	BL
	CX	CH	CL
	DX	DH	DL
	SI		
	DI		
Pointer	SP		
	BP		

- 8088/8086 đến 80286 : 16 bits
- 80386 trở lên: 32 bits EAX, EBX, ECX, EDX
- Y86-64: 64 bits RAX, RBX, RCX, RDX

Register Files- Segment registers

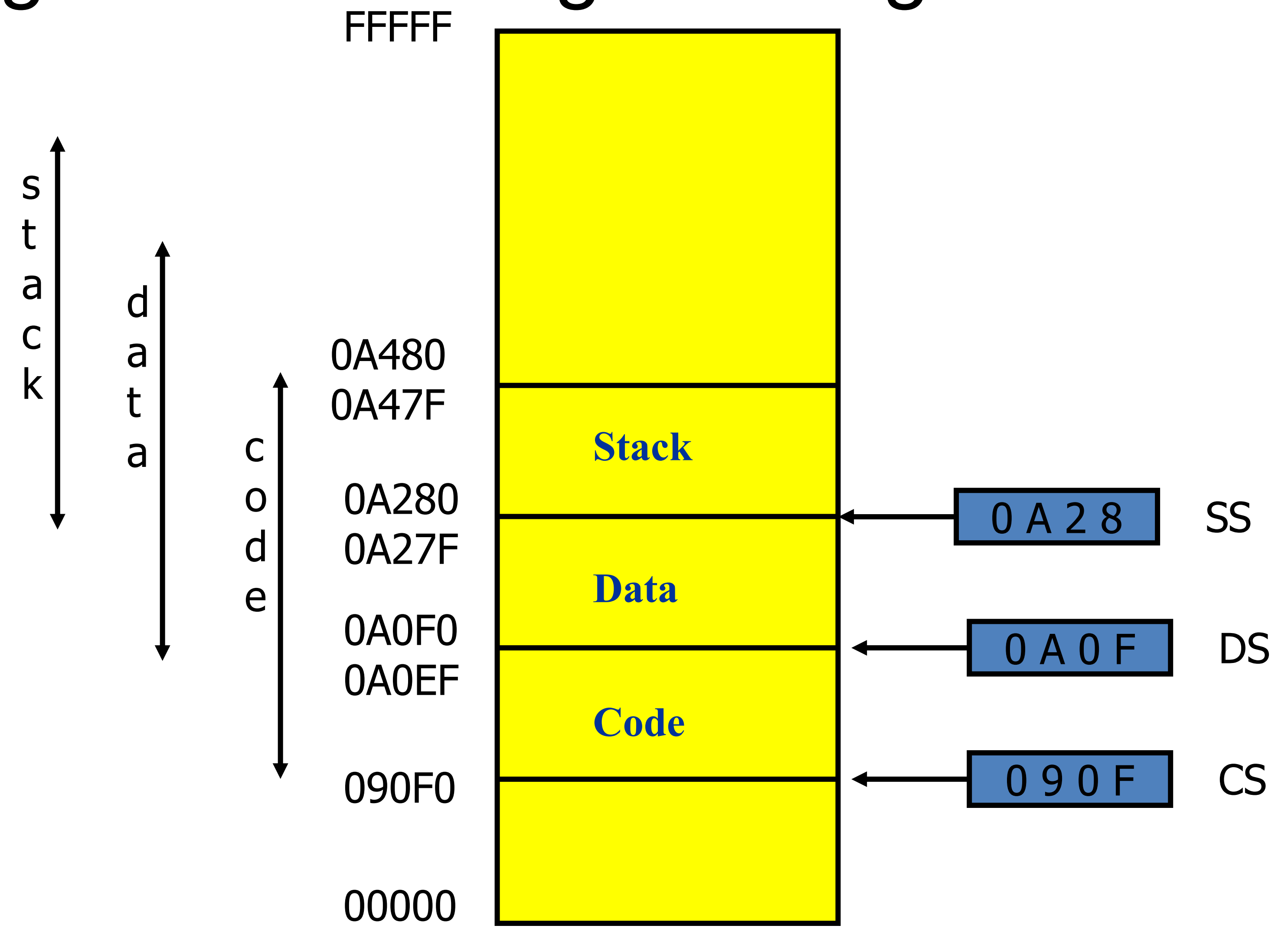
- 8088/8086 đến 80286 : 16 bits
- 80386 trở lên: 32 bits EAX, EBX, ECX, EDX
- Y86-64: 64 bits RAX, RBX, RCX, RDX



fit@hcmus

Register Files- Segment registers

- Nested segments: there is an overlap between the data segment, code segment and stack segment

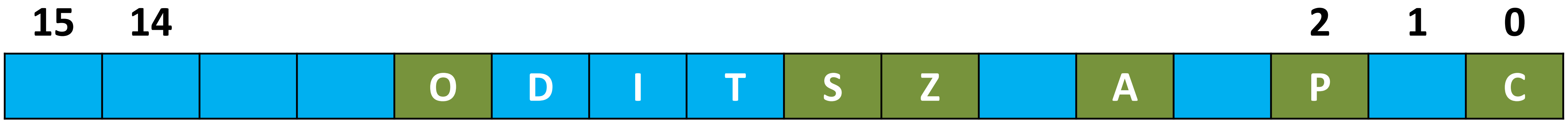


fit@hcmus

Pointer register and Indexed

Segment	Offset	Meaning
CS	IP	Instruction address
SS	SP hoặc BP	Stack address
DS	BX, DI, SI, số 8 bit hoặc số 16 bit	Data segment address
ES	DI	Destination string address

Flag register



6 bits are used to be status flags:

- C/CF (carry flag)): CF=1
- P/PF (parity flag): PF=1 (0) when the number of 1's bit in the result is even (odd)
- A/AF (auxilary carry flag): extended carry flag
- Z/ZF (zero flag): ZF=1 when the result is 0
- S/SF (Sign flag): SF=1 when the result is less than 0
- O/OF (Overflow flag): overflow detected in signed number computation

fit@hcmus

Flag register

15	14												2	1	0
				O	D	I	T	S	Z		A		P		C

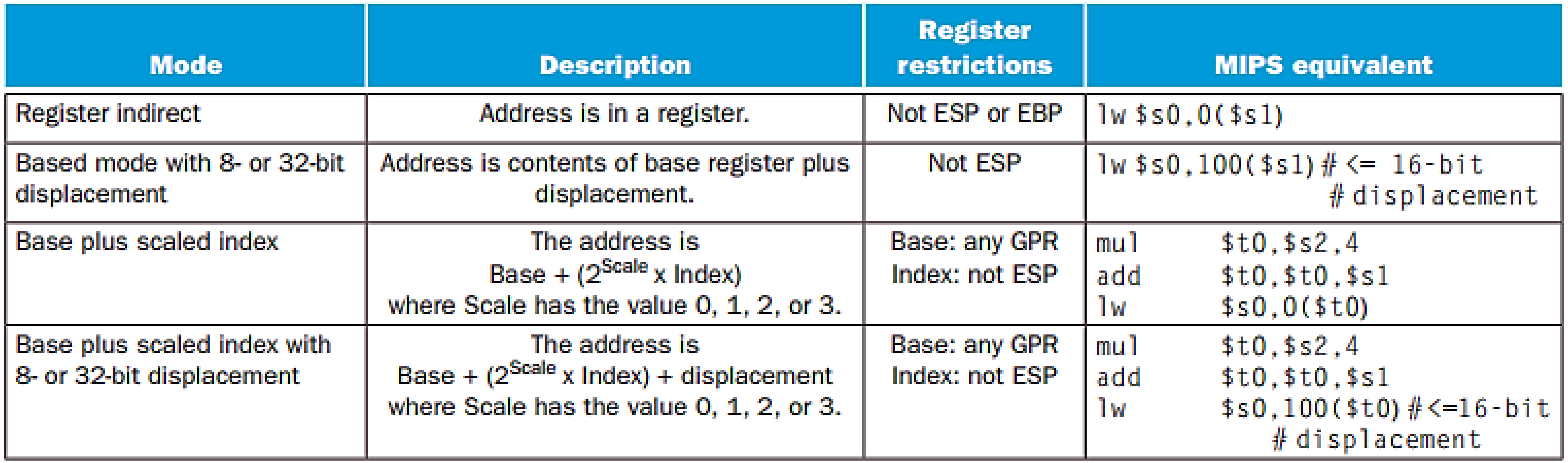
3 bits are used to be control flags:

- T/TF (trap flag)):): used for on chip debugging, TF=1 CPU will work in a single step mode. Generate an interrupt after each instruction
- I/IF (Interrupt enable flag): I = 1, CPU will recognize the interrupts from peripherals. For I = 0, the interrupts will be ignored
- D/DF (direction flag: D=1 the string will be accessed from higher memory address to lower memory address, and if D = 0, it will do the reverse

Data Addressing Mode

- Immediate
- Direct
- Indirect
- Register Direct
- Register Indirect
- Relative
- Indexed

Data Addressing Mode



Mode	Description	Register restrictions	MIPS equivalent
Register indirect	Address is in a register.	Not ESP or EBP	`lw $s0,0($s1)`
Based mode with 8- or 32-bit displacement	Address is contents of base register plus displacement.	Not ESP	`lw $s0,100($s1) # <= 16-bit # displacement`
Base plus scaled index	The address is Base + (2^{Scale} x Index) where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4` `add $t0,$t0,$s1` `lw $s0,0($t0)`
Base plus scaled index with 8- or 32-bit displacement	The address is Base + (2^{Scale} x Index) + displacement where Scale has the value 0, 1, 2, or 3.	Base: any GPR Index: not ESP	`mul $t0,$s2,4` `add $t0,$t0,$s1` `lw $s0,100($t0) #<=16-bit # displacement`

cdio 4.0
fit@hcmus

Data Addressing Mode

- Assume the following are stored as an indicated memory address and register

Address	Value	Register	Value
0x100	0xFF	%rax	0x100
0x104	0xAB	%rcx	0x1
0x108	0x13	%rdx	0x3
0x10C	0x11		

- Fill in the following table showing the value for indicated operands:

Operand	Value
%rax	
0x104	
$0x108	
(%rax)	
4(%rax)	
9(%rax,%rdx)	
260(%rcx,%rdx)	
0xFC(,%rcx,4)	
(%rax,%rdx,4)	

fit@hcmus

Data Addressing Mode

- Assume the following are stored as an indicated memory address and register

Address	Value	Register	Value
0x100	0xFF	%rax	0x100
0x104	0xAB	%rcx	0x1
0x108	0x13	%rdx	0x3
0x10C	0x11		

- Fill in the following table showing the value for indicated operands: (Solutions)

Operand	Value	Comment
%rax	0x100	Register
0x104	0xAB	Absolute address
$0x108	0x108	Immediate
(%rax)	0xFF	Address 0x100
4(%rax)	0xAB	Address 0x104
9(%rax,%rdx)	0x11	Address 0x10C
260(%rcx,%rdx)	0x13	Address 0x108
0xFC(,%rcx,4)	0xFF	Address 0x100
(%rax,%rdx,4)	0x11	Address 0x10C

cdio 4.0
fit@hcmus

OPERATIONS

- ☐ Data movement instructions
- ☐ Arithmetic and Logic instructions
- ☐ Control flow
- ☐ String instructions (include string move & compare)

fit@hcmus

Data movement instructions

- MOV: The mov instruction copies the data item referred to by its second operand into the location referred to by its first operand

```
Syntax:
mov <reg>,<reg>
mov <reg>,<mem>
mov <mem>,<reg>
mov <reg>,<const>
mov <mem>,<const>
```

```
Example:
copy the value in %bx into %ax
mov %AX, %BX
store the value 5 into the byte
at location var
mov byte ptr[var], 5
```

Data movement instructions

- PUSH: places its operand onto the top of the hardware supported stack in memory.

```
Syntax
push <reg32>
push <mem>
push <con32>
```

```
Example:
Push %eax on the stack
push %EAX
push the 4 bytes at
address var onto the stack
push [var]
```

Data movement instructions

- POP: removes the 4-byte data element from the top of the hardware-supported stack into the specified operand.

```
Syntax              Example:
pop <reg32>         pop the top element of the stack into EDI
pop <mem>           pop %EDI
                    pop the top element of the stack into
                    memory at the four bytes starting at
                    location EBX
                    push [%EBX]
```

fit@hcmus

Data movement instructions

- LEA: Load effective address.

```
Syntax                 Example:
lea <reg32> <mem>      the quantity EBX+4*ESI is placed in EDI
                       lea edi, [ebx+4*esi]
                       the value in var is placed in EAX
                       lea eax, [var]
                       the value val is placed in EAX
                       lea eax, [val]
```

cdio 4.0
fit@hcmus

Arithmetic and Logic instructions

- ADD: adds together its two operands, storing the result in its first operand.

```
Syntax
add <reg>,<reg>
add <reg>,<mem>
add <mem>,<reg>
add <reg>,<con>
add <mem>,<con>
```

```
Example:
EAX ← EAX + 5
add %eax, 5
add 5 to the single byte stored at
memory address var
add BYTE PTR[var], 5
```

fit@hcmus

Arithmetic and Logic instructions

- SUB: adds together its two operands, storing the result in its first operand.

```
Syntax                   Example:
sub <reg>,<reg>          AL ← AL - AH
sub <reg>,<mem>          sub %AL, %AH
sub <mem>,<reg>          subtract 5 from the value stored at %EAX
sub <reg>,<con>          sub %EAX, 5
sub <mem>,<con>
```

Arithmetic and Logic instructions

- INC/DEC: increments/ decrements the contents of its operand by one.

```
Syntax
inc <reg>
inc <mem>
dec <reg>
dec <mem>
```

```
Example:
add one to the 32-bit integer stored at location var
inc DWORD PTR [var]
subtract 1 from the contents of %EAX
dec %EAX
```

fit@hcmus

Arithmetic and Logic instructions

- iMUL: two basic formats: two-operand (first two syntax listings above) and three-operand (last two syntax listings above)

```
Syntax
imul <reg32>,<reg32>
imul <reg32>,<mem>
imul <reg32>,<reg32>,<con>
imul <reg32>,<mem>,<con>
```

```
Example:
multiply the contents of EAX by
the 32-bit contents of the
memory location var. Store the
result in EAX
imul %EAX, [var]
%ESI ← %EDI * 25
imul %ESI, %EDI, 25
```

Arithmetic and Logic instructions

- iDIV: divides the contents of the 64 bit integer EDX:EAX by the specified operand value. The quotient result of the division is stored into EAX, while the remainder is placed in EDX

```
Syntax
idiv <reg32>
idiv <mem>
```

```
Example:
divide the contents of EDX:EAX by the
contents of EBX
idiv %BX
divide the contents of EDX:EAX by the
32-bit value stored at memory
location var.
idiv DWORD PTR [var]
```

Arithmetic and Logic instructions

- CMP: Compare the values of the two specified operands, setting the condition codes in the machine status word appropriately (based on flag register)
  - Đích = nguồn : CF=0 ZF=1
  - Đích> nguồn : CF=0 ZF=0
  - Đích < nguồn : CF=1 ZF=0

```
Syntax
cmp <reg>,<reg>
cmp <reg>,<mem>
cmp <mem>,<reg>
cmp <reg>,<con>
```

```
Example:
If the 4 bytes stored at location var are
equal to the 4-byte integer constant 3,
jump to the location labeled loop
cmp DWORD PTR [var], 3
jeq loop
```

cdio 4.0
fit@hcmus

Arithmetic and Logic instructions

- AND, OR, XOR: Bitwise logical and, or and exclusive or. Placing the result in the first operand location

```
Syntax
opcode <reg>,<reg>
opcode <reg>,<mem>
opcode <mem>,<reg>
opcode <reg>,<con>
opcode <mem>,<con>
```

```
Example:
clear all but the last 4 bits of EAX
and %EAX, 0fH
set the contents of EDX to zero
xor %EDX, %EDX
```

cdio 4.0
fit@hcmus

Arithmetic and Logic instructions

- SHL, SHR: shift the bits in their first operand's contents left and right, padding the resulting empty bit positions with zeros

```
Syntax
opcode <reg>,<con8>
opcode <mem>,<con8>
opcode <reg>,<cl>
opcode <mem>,<cl>
```

```
Example:
Multiply the value of EAX by 2 (if
the most significant bit is 0)
shl %EAX, 1
Store in EBX the floor of result of
dividing the value of EBX by 2^n where
n is the value in CL
shr %EBX, %CL
```

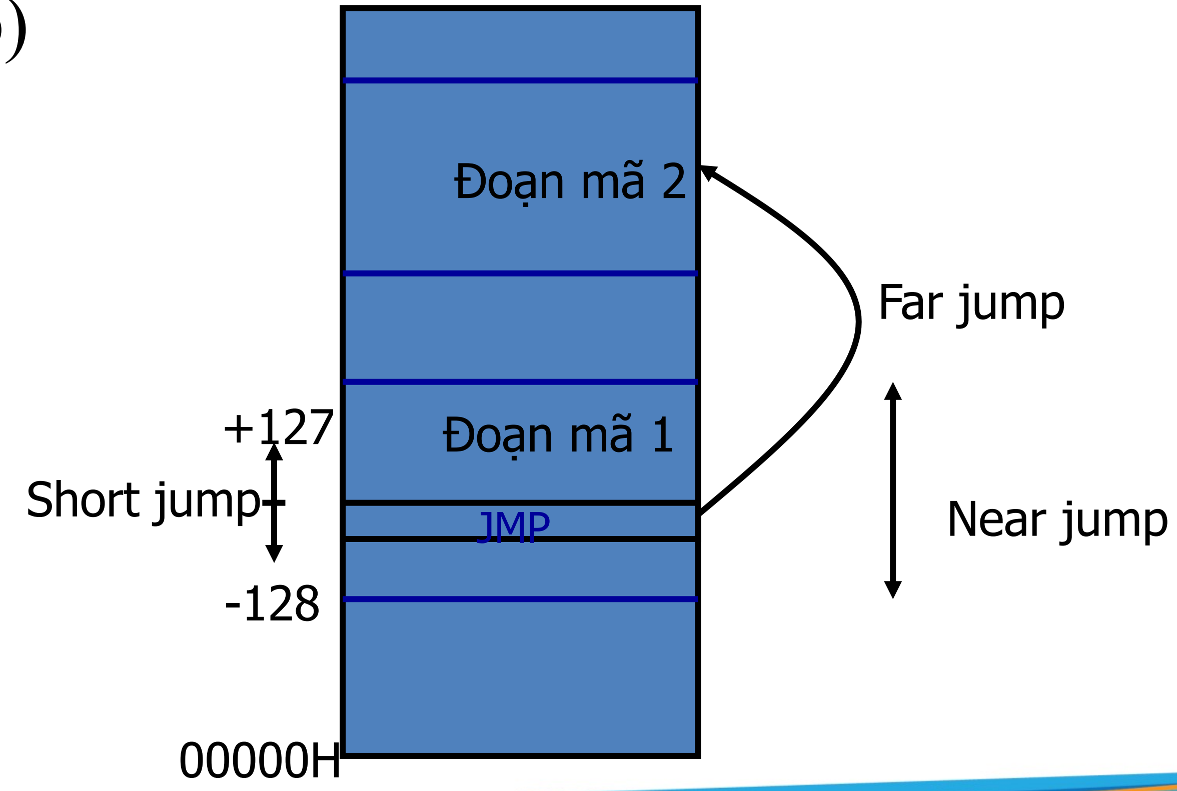
fit@hcmus

Control flow instructions

- JMP: transfers program control flow to the instruction at the memory location indicated by the operand
- 3 types of JMP instruction: JUMP SHORT(short jump), JUMP NEAR (near jump), JUMP FAR (far jump)

```
Syntax
jmp <Label>

Example:
Jump to the instruction
named "BEGIN"
Jmp BEGIN
```



fit@hcmus

Control flow instructions

- JE, JNE, JZ, JG, JGE, JL, JLE: Conditional jump
- Based on the status of a set of condition codes that are stored in a special register called the machine status word

```
Syntax
opcode <Label>
```

```
Example:
Jump to the instruction named "DONE"
if the condition satisfies
cmp %EAX, %EBX
jle done
```

fit@hcmus

Control flow instructions

- LOOP, LOOPE/LOOPZ, LOOPNE/LOOPNZ: is a combination instruction of DEC CX and JNZ

```
Syntax
<Label:>
     Task
Loop <Label>
```

```
Example:
Repeat when CX != 0. Decrements CX
after each loop
XOR AL, AL
MOV CX, 16
myloop:
     INC AL
     LOOP myloop
```

String instructions

- MOVS, MOVSB, MOVSW: copy from the string source (located in data segment) to destination (located in extra segment) by increment ESI and EDI; may be repeated

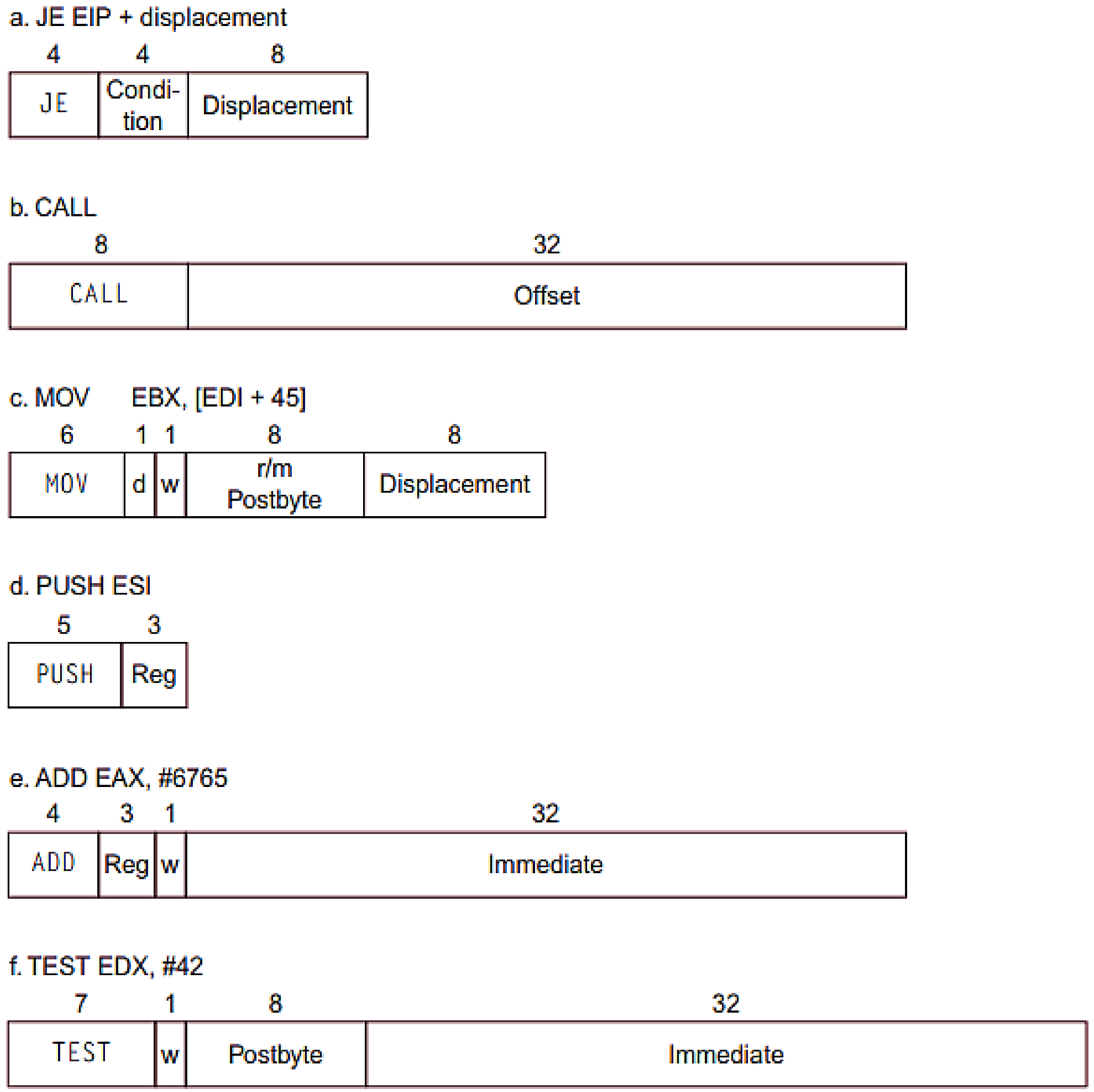
```
Example:
move a string of length 4 bytes from source to destination
MOV SI, SRC
MOV DI, DST
MOV CX, 04H
CLD; Clear the direction flag
REP MOVSB
```

Instruction Encoding

Variable length encoding

- Postfix bytes specify addressing mode
- Prefix bytes modify operation

  Operand length, repetition, locking, …



cdio 4.0
fit@hcmus

Interrupt I/O

- Generated by the software
- 2 types:
  - Use IN, OUT command to swap with out of external devices
  - Use DOS and BIOS interrupt server subroutines (independent on system)

Interrupt I/O

- Interrupt 21h of DOS

Code (AH)	Meaning	Arguments	Result
1	Read a character		AL
2	Print a character	DL = ascii code of the character	
9	Print string with null character in the end	DX = address of the string	
4CH	End of .exe program		