EEE 204 Microcomputer Organization

Asst. Prof. Dr. Seydi Kaçmaz

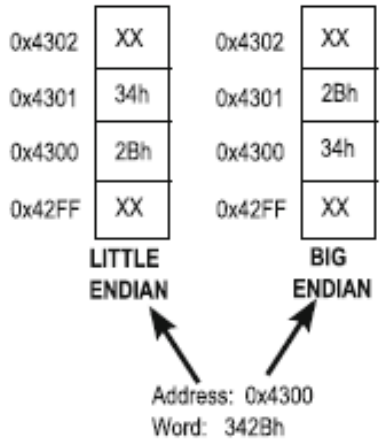
CHAPTER 3

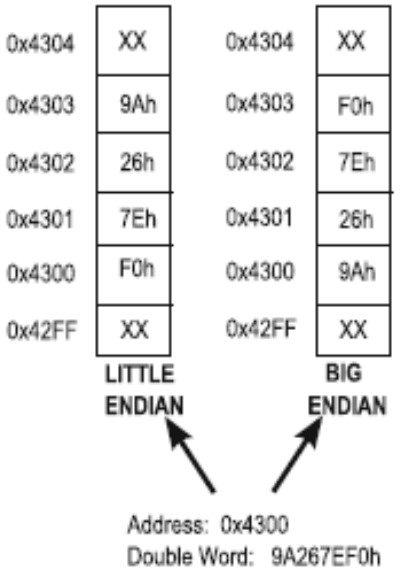
3.5 MSP430 Memory Organization

Big Endian: In the big endian convention, data is stored with the most significant byte in the lowest address and the least significant byte in the highest address.

Little Endian: In the little endian convention, data is stored with the least significant byte in the lowest address and the most significant byte in the highest address.

In this example, a 16-bit of data (342BH) is intended to be saved to the memory location 4300H in little and big endian conventions



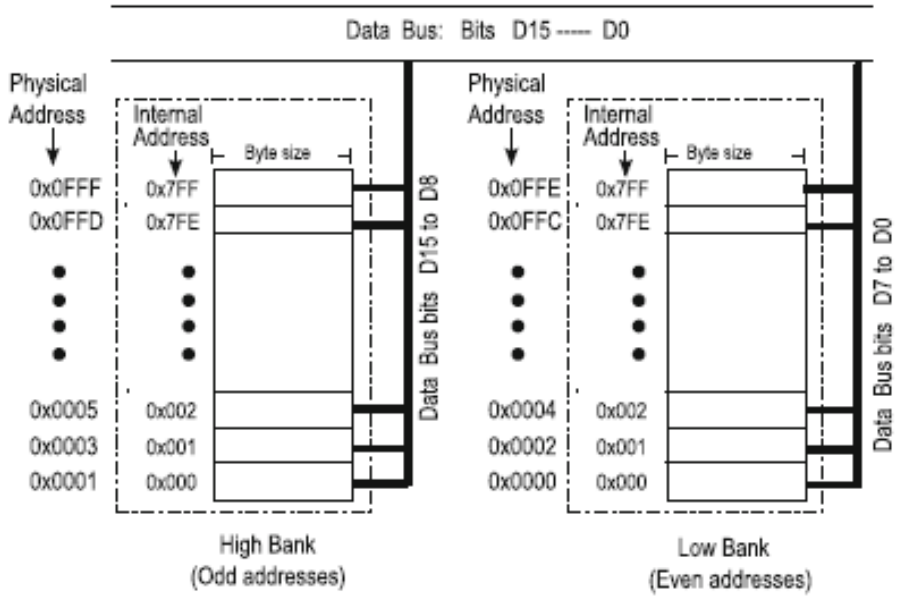


In this example, a 32-bit of data (9A267EF0H) is intended to be saved to the memory location 4300H in little and big endian conventions

3.5 MSP430 Memory Organization

Most hardware memory words nowadays are one byte length. Each memory word has an address attached to it, referred to as its **physical address,** which is encoded by the group of the MCU address bus bits. Memory blocks one byte length are called **banks**.

For data buses wider than a byte, two or more banks are needed to connect all the data bus lines, as illustrated in Figure for a 16-bit data bus. In this figure, physical address refers to the address seen by the CPU with the address bus, while the internal addresses are the addresses proper to the bank.



3.5 MSP430 Memory Organization

Example: The debugger of a certain microcontroller presents memory information in chunks of words in the list form shown below, where the first column is the address of the word in the second column. Following the debuggers' conventions, all numbers are in hex system**. If more than one word is on the line, the address is for the first word only.** Assuming that all data is effectively 16-bit wide, break the information into bytes with the respective address, (a) assuming little endian convention, and (b) assuming big endian convention.

F81E: E0F2 0041 0021

F824: 403F 5000

F828: 831F

Solution: *According to instructions, F81E is the address of word E0F2, F820 that of the second word 0041 and so on. With this in mind, we can break the memory information in byte chunks as follows:*

Little endian		Big endian	
F81E:F2	F824: 3F	F81E: E0	F824: 40
F81F: E0	F825: 40	F81F: F2	F825: 3F
F820: 41	F826: 00	F820: 00	F826: 50
F821: 00	F827: 50	F821: 41	F827: 00
F822: 21	F828: 1F	F822: 00	F828: 83
F823: 00	F829: 83	F823: 21	F829: 1F

3.6.1 Anatomy of an I/O Interface

Control Registers: Allow for configuring the operation of the device and the interface itself. One or several control registers can be provided depending on the complexity of the interface. Sometimes this type of register is called Mode or Configuration Register. (Special Purpose Registers)

Status Register: Allow for inquiring about the device and interface status. Flags inside these registers indicate specific conditions such as device ready, error, or other condition. (Special Purpose Registers)

Data Registers: Allow for exchanging data with the device itself. Unidirectional devices might have only one data register (Data-in for input devices or Data-out for output devices). Bi-directional I/O interfaces include both types. (General Purpose Registers)

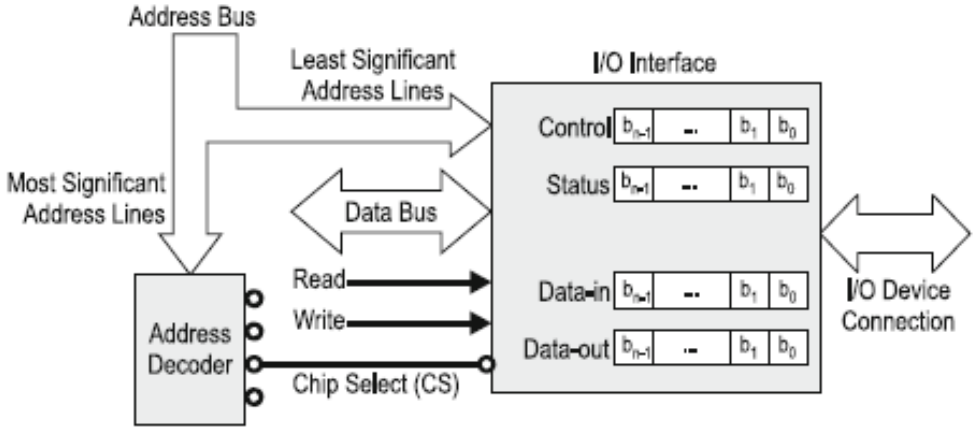


Fig. Anatomy of an I/O interface

3.6.1 Anatomy of an I/O Interface

Direction Register (PxDIR): Selects IN or OUT direction function for the pin, *with 1 for output direction and 0 for input direction.*

Input Register (PxIN): This is a read-only register. The value changes automatically when the input itself changes.

Output Register (PxOUT): to write signal to output. This is a read-and-write register.

Function Select Register (PxSEL): Used to select between I/O port or peripheral module function. With PxSEL.n = 0, pin Px.n operates as an I/O pin port; with PxSEL.n = 1, as a module pin (other function).

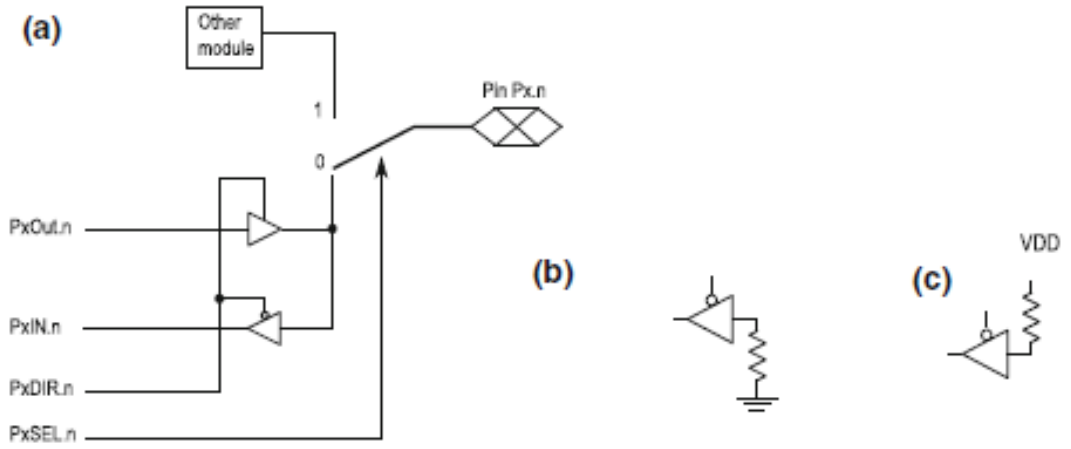


Fig. Basic IO Pin hardware configuration:
a) Basic I/O register's functions
b) Pull-down resistor for inputs
c) Pull-up resistors

3.6.1 Anatomy of an I/O Interface

Let's say for example, pins 7 to 3 of port 6 are to be used as output pins, and pin 2 for operating the module, then we should use appropriate CPU instructions to write 0xF8 in the P6DIR register and 04h in the P6SEL register.

- P6DIR=11111000=0xF8, P6SEL=00000100= 04h

Therefore, in 6^{th} port, B7,…B3 are output pins while B2, B1 and B0 are input pins. Moreover, all pins of 6^{th} port except B2 are GPIO (General Purpose Input/Output) pins

Since the input port goes through a three-state buffer, it is not advisable to leave it floating when the pin operates in input mode. It is necessary to connect a pull-up or a pull-down resistor.

Additional configuration registers might be included as part of an I/O port. Examples include registers to configure interrupt capabilities in the port, or to use internal pull-up/pull-down resistors, and other functions.

3.7.1 Register Transfer Notation (RTN)

It is important for the programmer to have a notation available for operations in MCU environments, independent of the specific MCU architecture but taking into consideration the features of the systems. One such notation is the **Register Transfer Notation (RTN).**

After executing an instruction, the contents of a register or cells in memory may be written upon with a new datum. In this case, the register or cell being modified is called **destination**. The **source** causing the change at destination may be a **datum**(one piece of data) being transferred (copied) or the result of an operation. This process is denoted in abstract form as

destination ← *source* OR *source* → *destination*

3.7.1 Register Transfer Notation

The notation in programmer's model for the different operands that may be used in RTN are as follows:

1. Constants: These are expressed by their value or by a predefined constant name, for example 24, 0xF230, MyConstant. Constants cannot be used in destination.

2. Registers: These are referred to by their name. If it is in abstract form without reference to a particular CPU, it is customary to use Rn, where n is a number or letters. (Remember for MSP430, they are only numbers. $R_{0,\ldots\ldots}R_{15}$.)

3. Memory and I/O: These are referred to by the address in parenthesis, as in(0x345E), which means "The data in memory at address 0x345E." Notice that it is data address, not physical address. If the address information is contained in register Rn we write (Rn), meaning by that "The data in memory at address given by Rn". We also say that the register points to the data.

3.7.1 Register Transfer Notation

Example: The following transactions illustrate RTN for memory operands. For these examples, let us assume word-size data at addresses before each transaction as [0246h] = 32AFh and [028Ch] = 1B82h. Moreover, let us assume little endian storage.

RTN	Meaning	Result
(0246h) ← 028Ch	Store 028Ch at data address 0246h	[0246h] = 028Ch
Word(0246h) ← (028Ch)	Copy at memory location 0246h the word at memory address 028Ch	[0246h] = IB82h
Byte(0246h) ← (028Ch)	Copy at memory location 0246h the byte at memory address 028Ch	[0246h] = 3282h
(0246h) ← (0246h) + 3847h	Add 3847h to the current contents of memory location 0246h	[0246h] = 6AF6h
Byte(028Dh) ← (028Dh) − (0247h)	Subtract the byte memory location 0247h from the byte at memory address 028Dh	[028Ch] = E982h

3.7.1 Register Transfer Notation

Example: Assume two 16-bit registers, R6 and R7, with contents R6 = 4AB2h and R7 = 354Fh, respectively. Moreover, assume words at addresses [4AB2h] = 02ACh, [4C26h] = 94DFh and [4AB8h] = 3F2Ch. Assume little endian convention when necessary. The following examples illustrate more RTN expressions:

RTN	Meaning	Result
R6 ← 3FA0h	Load R6 with 3FA0h	R6 = 3FA0h
R7 ← R6	Copy in R7 the current contents of R6	R7 = 4AB2h
R7 ← (R6)	Copy in R7 the word whose memory address is stored in R6. Also: Copy in R7 the word in memory pointed at by R6.	R7 = 02ACh
byte(4C26h)← byte(4C26h) − (R6)	Subtract from the byte at address 4C26h the byte whose address is given by R6	[4C26h] = 9433h
(R6) ← (R6) + R7	Add the contents of R7 to the word pointed at by R6.	[4AB2h] = 37FBh
R7 ← (R6 + 6h)	Copy in R7 the memory word whose address is given by R6 + 6h	R7 = 3F2Ch
byte(R6) ← (R6) − 0x92	subtract 0x92 from the current memory byte at address given by contents of R6.	[4AB2h] = 021Ah

The MOV Instruction

- **MOV** is the primary instruction to **copy (not cut)** information within the computer system.
- **src (Source),** the location of where the information is to be copied **from**
- **dst (Destination),** the location of where the information is to be copied **to**
- The **src** and **dst** can have operands as dictated in Addressing Modes
- It performs both 16-bit and 8-bit operations, dictated by the extensions .w and .b. If no extension is used, the instruction is assumed to be a 16-bit operation.

```
mov.w src, dst; .w is optional
```

.w tells the assembler this is a 16-bit operation

.w treats the src and dst as 16-bit words.

```
mov.b src, dst
```

.w tells the assembler this is a 8-bit operation

.w treats the src and dst as 8-bit words.

3.7.5 Addressing Modes

Addressing modes can be defined as the way in which an operand is specified within an instruction so as to indicate where to find the data with which the operation is executed.

In general, the data to be used or stored in a transfer or in an arithmetic or logic instruction can be located in only one of the following possible places:

1. It may be explicitly given,
2. It may be stored in a CPU register,
3. It may be stored at a memory location, or
4. It may be stored in an I/O port or peripheral register.

3.7.5 Addressing Modes

Seven addressing modes for the source operand and four addressing modes for the destination operand for MSP430. **Regardless of the type of the addressing modes, registers can be in the place of source or destination!**

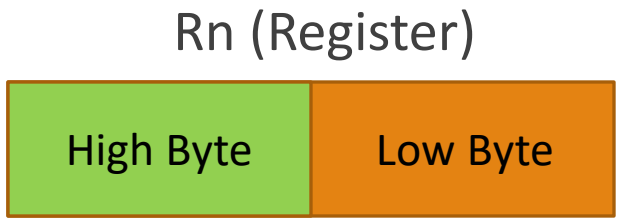
Source	Destination	Addressing Mode	Syntax	Description
YES	YES	Register	`Rn`	The operands are register names (i.e, PC, R4)
YES	YES	Indexed	`X(Rn)`	(Rn + X) points to the operand.
YES	YES	Symbolic	`ADDR`	The word (ADDR) is a label of the address of where information is to be accessed
YES	YES	Absolute	`&ADDR`	The word following the instruction contains the absolute address
YES	**NO**	Indirect Register	`@Rn`	Rn is used as a pointer to the operand.
YES	**NO**	Indirect Auto Increment	`@Rn+`	Rn holds the address of where information is to be accessed. After access, the register is incremented.
YES	**NO**	Immediate	`#N`	The operand N is a constant value to put into destination

Table. Addressing modes for MSP430

Register Mode

In register mode addressing, the operand for either the source and/or destination are CPU registers.

```
mov.w R4, R5      ;copy R4 into R5
mov.w R5, R6      ;copy R5 into R6
mov.b R7, R8      ;copy Low Byte of R7 into R8
mov.b R8, R9      ;copy Low Byte of R8 into R9
mov.w R10, R11    ;copy R10 into R11
mov.w R11, R12    ;copy R11 into R12
```



Indexed Mode

Indexed mode is similar to indirect register mode in that a register name is provided that holds the address of where to access the information. Index mode extends this functionality by allowing a numeric constant to be added to the contents of the register. This constant is called an ***offset,*** which is numeric and 16-bit signed number.

```
mov.w   0(R4), 8(R5)   ;copy content of the memory location addressed by R4 into the memory location addressed by R5+8

mov.w   2(R4), 10(R6)  ;copy content of the memory location addressed by R4+2 into the memory location addressed by R6+10

mov.w   3(R7),  8(R8);copy content of the memory location addressed by R7+3 into the memory location addressed by R8+8

mov.w  6(R8),  14(R9)  ;copy content of the memory location addressed by R8+6 into the memory location addressed by R9+14
```

Symbolic Mode

In symbolic mode, the address label is simply inserted in either the src or dst fields without any preceding indicator. Address label is used as the operand and represents the memory location. Symbolic mode supports labels for 16-bit addresses so that just like absolute mode. Label is a sequence of characters that represents a 16-bit address in memory

```
mov.w Const1, R4   ;copy content at address label Const1 into R4
mov.w R4, Var1     ;copy content of R4 into address label Var1
mov.w Const2, R5   ;copy content at address label Const2 into R5
mov.w R5, Var2     ;copy content of R5 into address label Var2
```

`Const1`, `Const2`, `Var1` and `Var2` are labels and they must be defined in the program.

Absolute Mode

In absolute mode addressing the src and/or dst is a 16-bit address value. This mode is the first to allow us to access the memory system of the MSP430. This mode only supports 16-bit addressing. The operand is a 16-bit address. This 16-bit address is the actual address that will be accessed. The term **absolute** means that the address provided is the **actual address** to access and not a variable name or label.

This allows us to move information from an address location into another address location but we can also mix and match. We can also move something from an address location into a register and vice versa. To denote absolute addressing, the value must be preceded by **&** (ampersand).

```
mov.w &2000h, R4   ;copy contents of 2000H address into R4
mov.w R4, &2004h   ;copy R4 into 2004h address
mov.w &2002h, R5   ;copy contents of 2002H address into R5
mov.w R5, &2006h   ;copy R5 into 2006h address
mov.w &2006h, &2002h ;copy contents of 2006H address into 2002H address
```

Indirect Register Mode

In indirect register mode, a CPU register is used to provide the address of where the information to be accessed is stored. **@ (at)** symbol is placed in front of the register name to indicate the use of indirect register name.

```
mov.w #2000h, R4  ;load 2000h into R4

mov.w @R4, R5 ;copy the content of memory location addressed
by the content of R4 into R5

mov.w @R5, R6 ;copy the content of memory location addressed
by the content of R5 into R6. Also remember the content of R5
comes from previous line
```

Indirect Auto Increment Mode

In indirect auto increment mode, the pointing register is automatically incremented after the completion of the instruction. The pointing register can be incremented by 1 or 2 depending on the type of the size of the operation dictated by .w and .b.

```
mov.w @R4+, R5 ;copy data that is addressed by R4 into R5 and R4=R4+2
mov.w @R4+, R6 ;copy data that is addressed by R4 into R6 and R4=R4+2
mov.w @R4+, R7 ;copy data that is addressed by R4 into R7 and R4=R4+2
mov.b @R4+, R8 ;copy data that is addressed by R4 into R8 and R4=R4+1
mov.b @R4+, R9 ;copy data that is addressed by R4 into R9 and R4=R4+1
mov.b @R4+, R10 ;copy data that is addressed by R4 into R10 and R4=R4+1
```

Immediate Mode

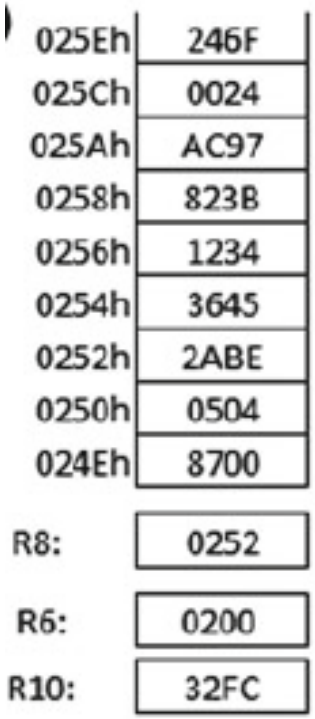
In this mode, the value of the operand Number is the datum. Immediate mode is reserved only for source operands, since a number cannot be changed by an operation. **# (pound)** symbol is placed in front of the constant number

```
mov.w #1234h, R4        ;load 1234h into R4
mov.w #0FACEh, R5       ;load FACEh into R5
mov.b #99h, R6          ;load 99h into R6
mov.b #0EEh, R7         ;load EEh into R7
mov.w #371, R8          ;load 371 decimal into R8
mov.b #10101010b, R9    ;load 10101010 binary into R9
mov.b #'B', R10         ;load ASCII code for B(42h) binary into R10
```

Not: If the number starts with a letter, a 0(zero) must be inserted before the number to avoid confusion. Otherwise, Assembler may treat the letter(s) as variable name or characters depending on the program.

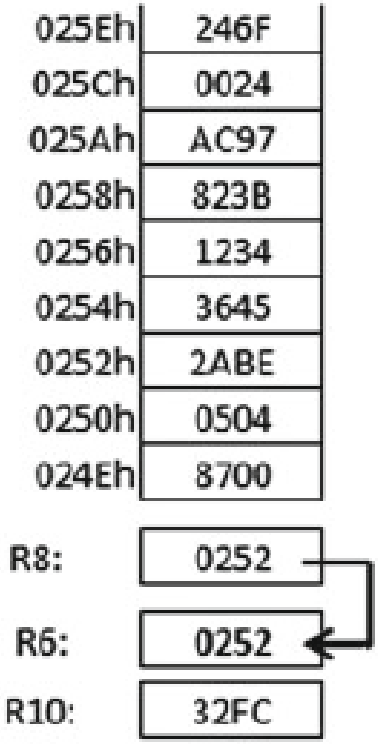
Examples

Before



`mov R8, R6`

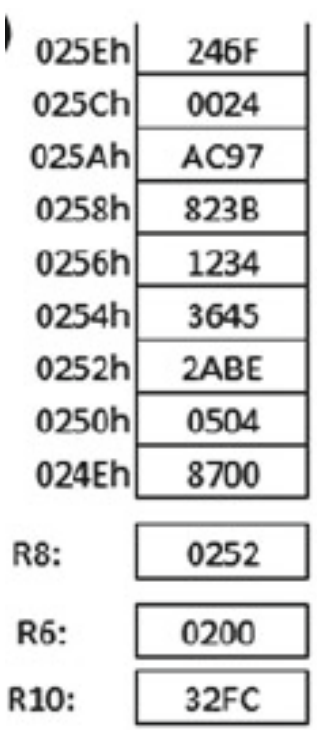
After



Register Mode

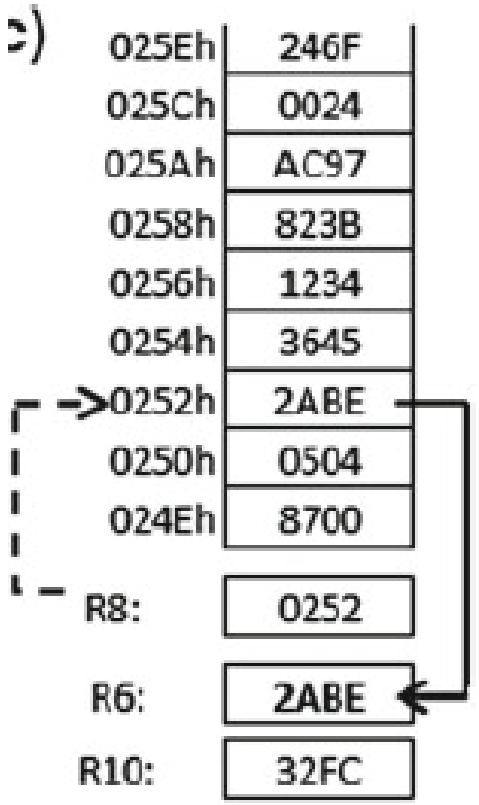
Examples

Before



`mov @R8, R6`

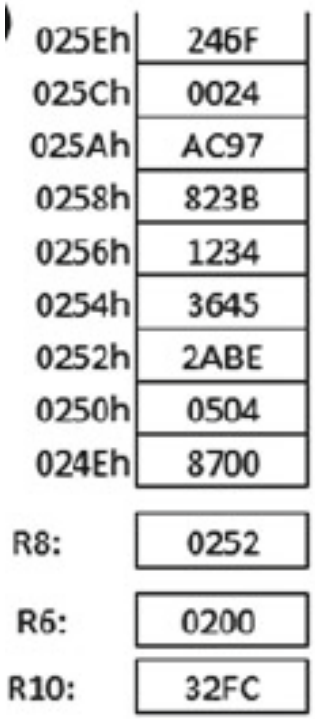
After



Indirect Register Mode

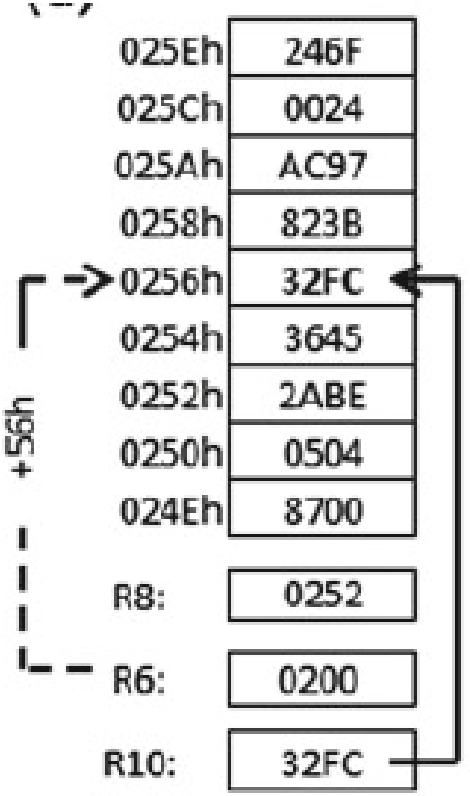
Examples

Before




```
mov R10, 56h(R6)
```

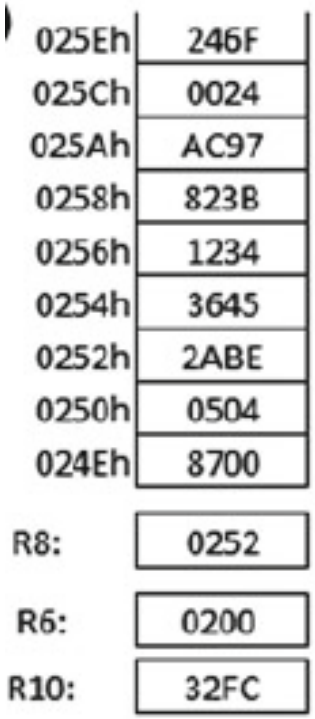
After

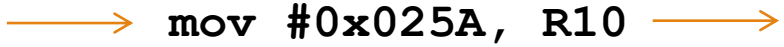


Indexed Mode

Examples

Before



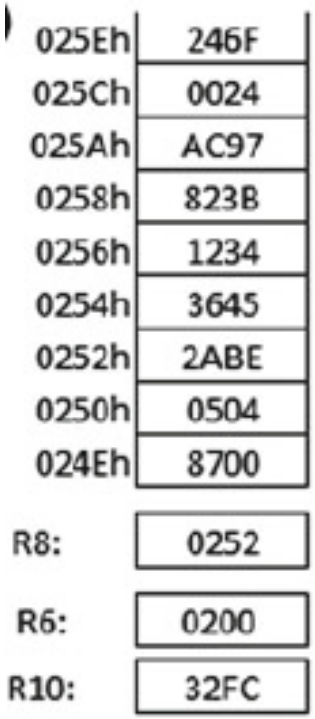


After

Immediate Mode

Examples

Before



→ `add R8, R6` →

After

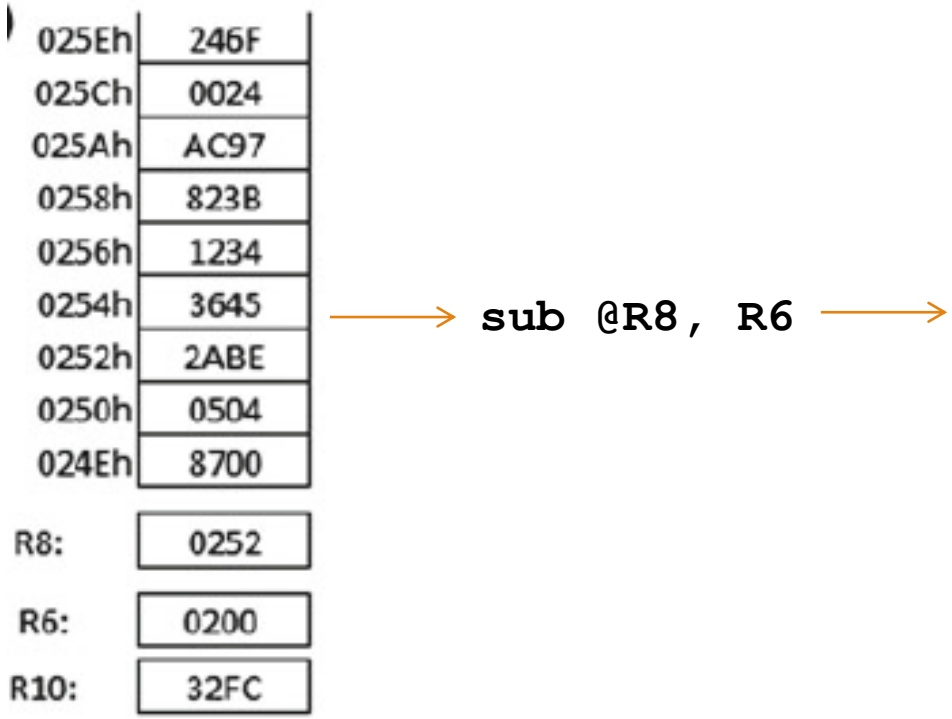
R6←R6 + R8 in RTN, means “add the contents of registers R8 and R6, and store the result in Register R6”.

Since 0200h + 0252h = 0452h, Then R6 = 0452h after this instruction.

Register Mode

Examples

Before



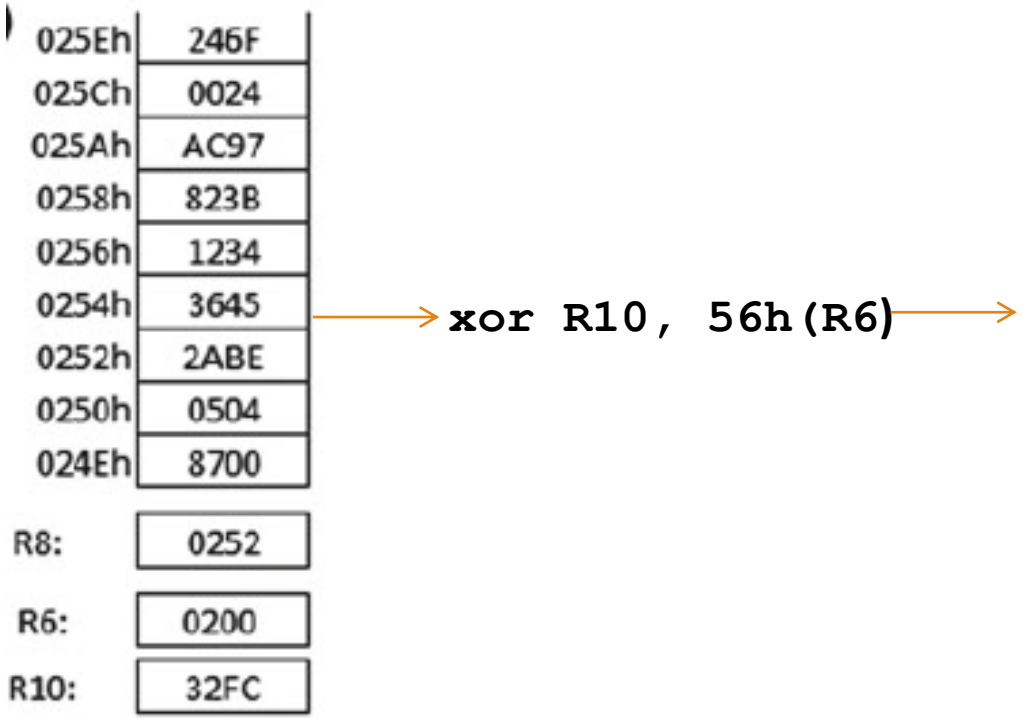
→ `sub @R8, R6` →

After

R6←R6 − (R8) in RTN, means “subtract from the contents of R6 the data in memory located at the address provided by register R8, and store result in Register R6”. Since R8=0252h, we search for address 0252h in the memory space where we see [0252h] = 2ABEh.
Now 0200h − 2ABEh = D742h.
Therefore, R6 = D742h after this instruction.
Indirect Register Mode

Examples

Before



After

(R6 + 56h) ← R10.xor.(R6 + 56h) in RTN, means "perform a bitwise XOR operation with the contents of register R10 and the content of the memory location whose address is found by adding 56h to the contents of register R6, and store the result in the same memory location".

Since R10 = 32FCh and (R6+56h)=(0256h)=1234h

32FCh XOR 1234h = 20C8h → (0256h)

Indexed Mode