

Try to Design and Implementation
of a PIC16 compatible microcontroller

Skill up course

PIC16 Microcontroller

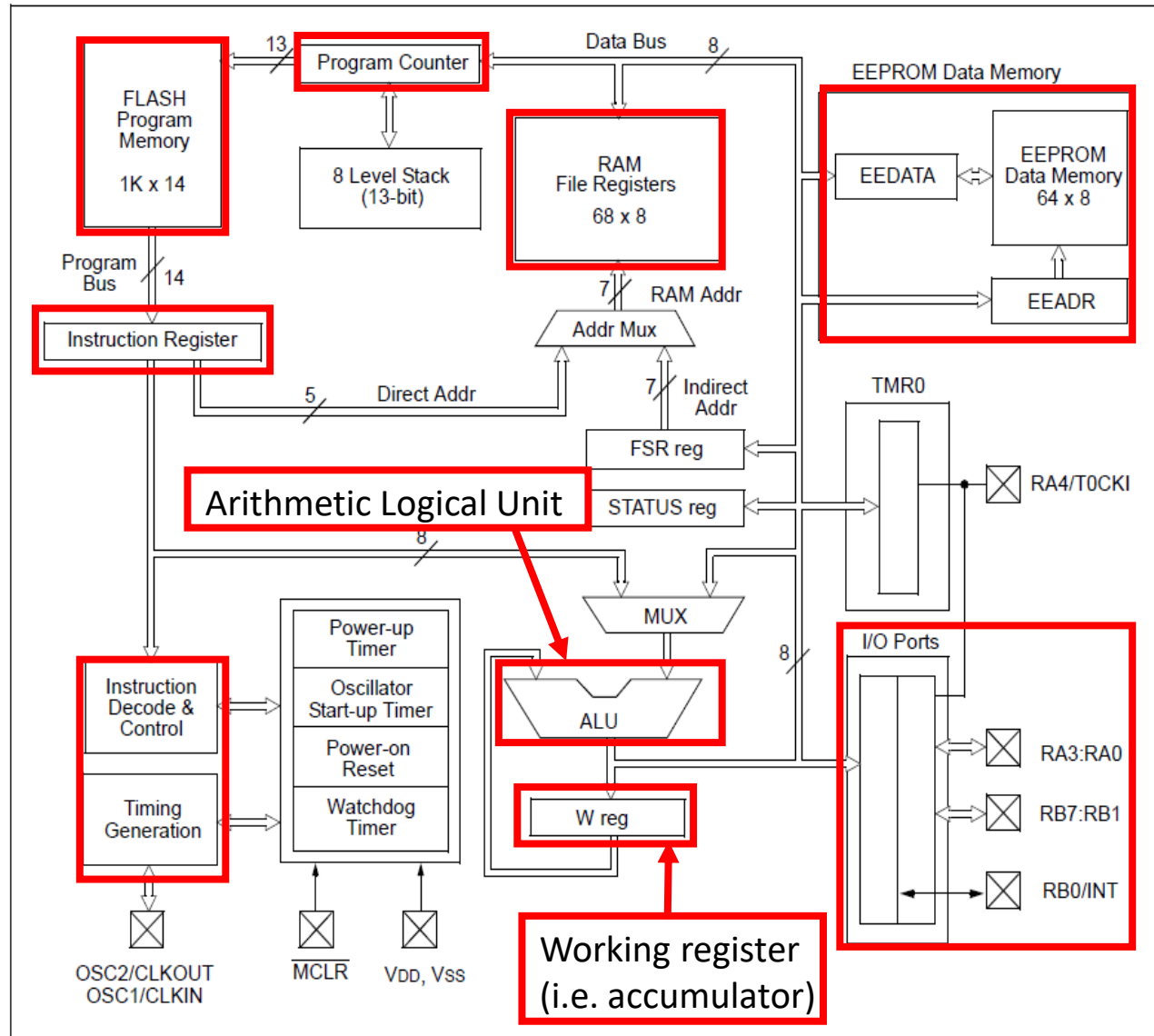
PIC16F84A by Microchip

Widely used for
electronics workshop.



Microcontroller:
It is one chip computer
includes of microprocessor,
memory and some
peripherals.
You can make several
controllers.

FIGURE 1-1: PIC16F84A BLOCK DIAGRAM



Sample program : Store a value into data memory

A constant value is stored into data memory combined with "movlw" and "movwf" instructions

movlw D'10' Store a constant 10₍₁₀₎ in instruction field to W register
movwf H'20' Store a value of W register into address 20₍₁₆₎ of data memory

mnemonic operand

Assembly Language:

It is a language for ease of understanding machine language.

Processor only recognizes binary value.

Program is also represented by binary value.

(Machine Language)

movlw D'10' 11_0000_0000_1010
movwf H'20' 00_0000_1010_0000

Machine language is generated by Assembler automatically.

Constants are represented by binary.

10₍₁₀₎ → 1010₍₂₎, 20₍₁₆₎ → 10000₍₂₎

Instruction register

· · 10₍₁₀₎

movlw

W

10₍₁₀₎

Address

20₍₁₆₎

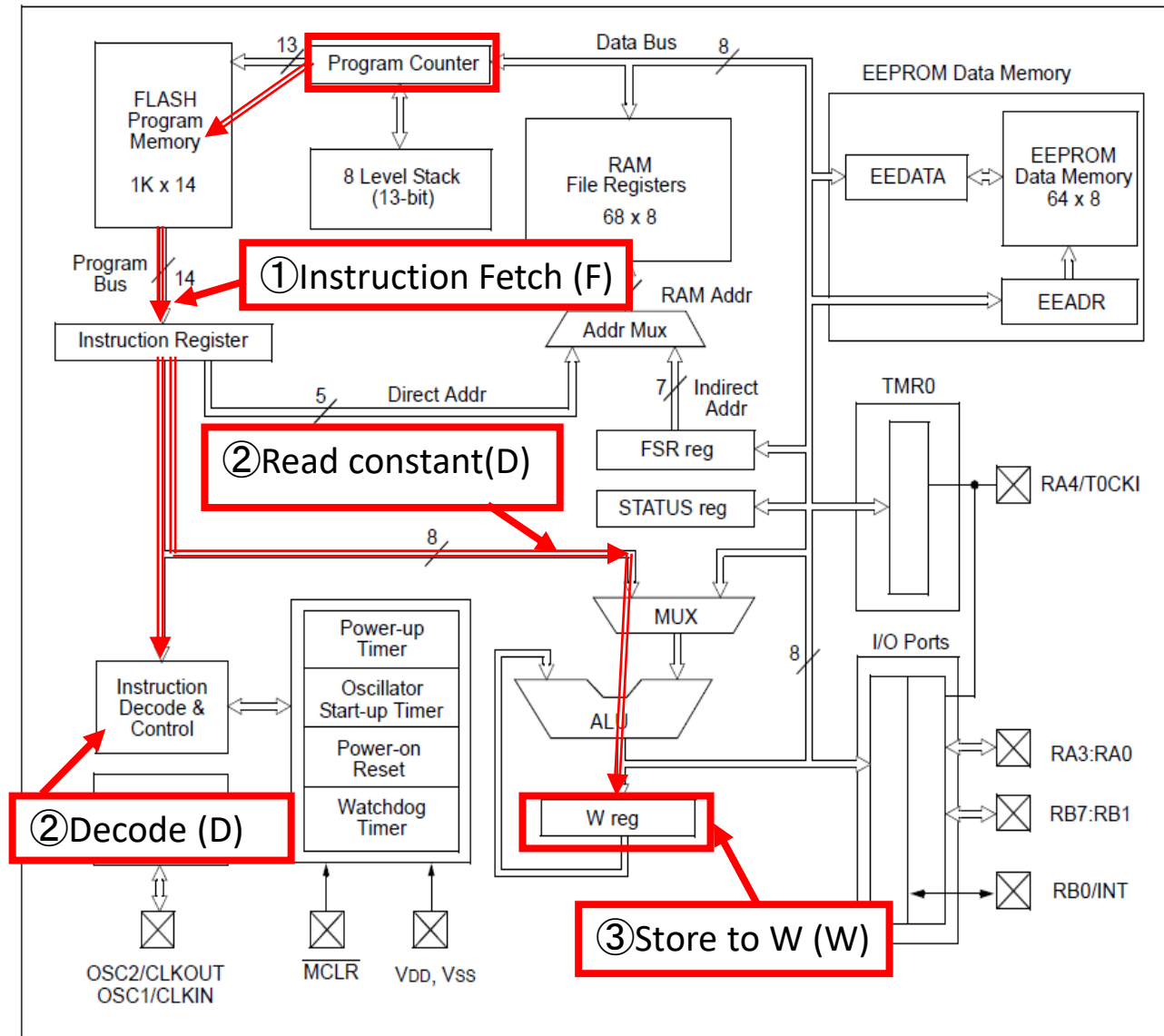
10₍₁₀₎

movwf

Data memory

Behavior of "movlw" instruction

FIGURE 1-1: PIC16F84A BLOCK DIAGRAM



Instruction behavior that
a constant in instruction
is stored to W register.

movlw constant

① Instruction pointed
out by program
counter is stored to
instruction register.

② Instruction is
decoded. And constant
in instruction is ready
to be read.

③ Store the constant to
W register.

Behavior of "movwf" instruction

Instruction behavior that
a value in W register is
stored into data memory.

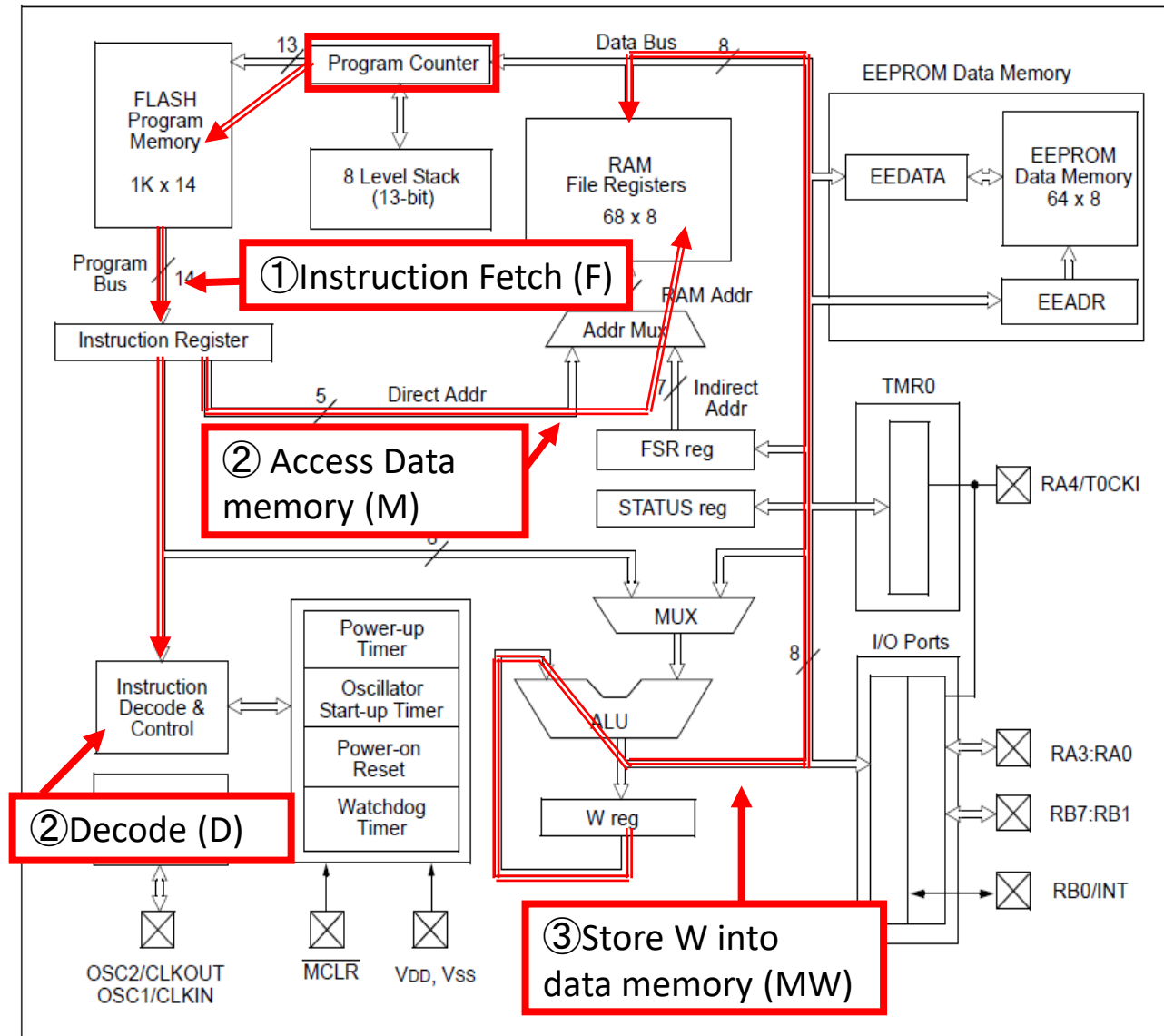
movwf Address

① Instruction pointed
out by program counter
is stored to instruction
register.

② Instruction is
decoded. And address
for accessing data
memory is ready.

③ W register value is
stored into data memory
through the
ALU

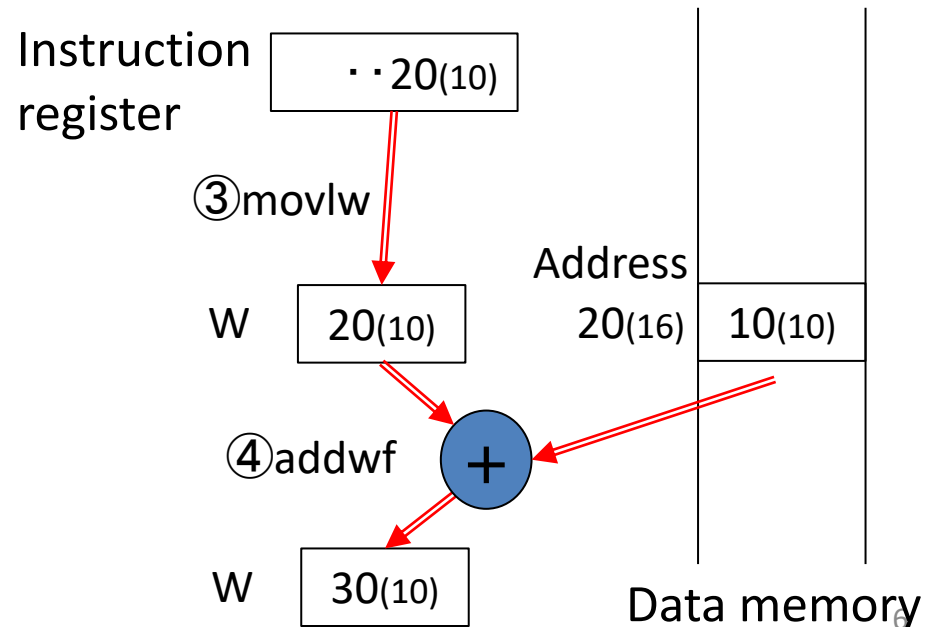
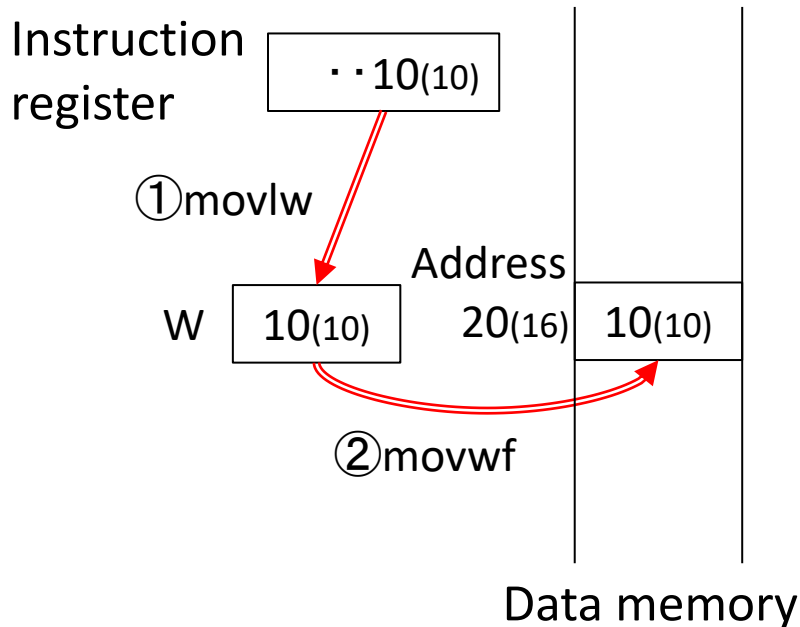
FIGURE 1-1: PIC16F84A BLOCK DIAGRAM



Sample Program : Addition

Add a value in data memory and W register by "addwf" instruction (Example: $10+20=30$)

- ① movlw D'10' Store a constant 10(10) in instruction field to W register
- ② movwf H'20' Store a value of W register into address 20(16) of data memory
- ③ movlw D'20' Store a constant 20(10) in instruction field to W register
- ④ addwf H'20',w Add W register and a value of data memory addressed by 20(20), and the result 20(10) is stored to W register



加算命令の動作例 (addwf Address,w)

Instruction behavior that a value in data memory and W register value are added and the result is stored to W register.

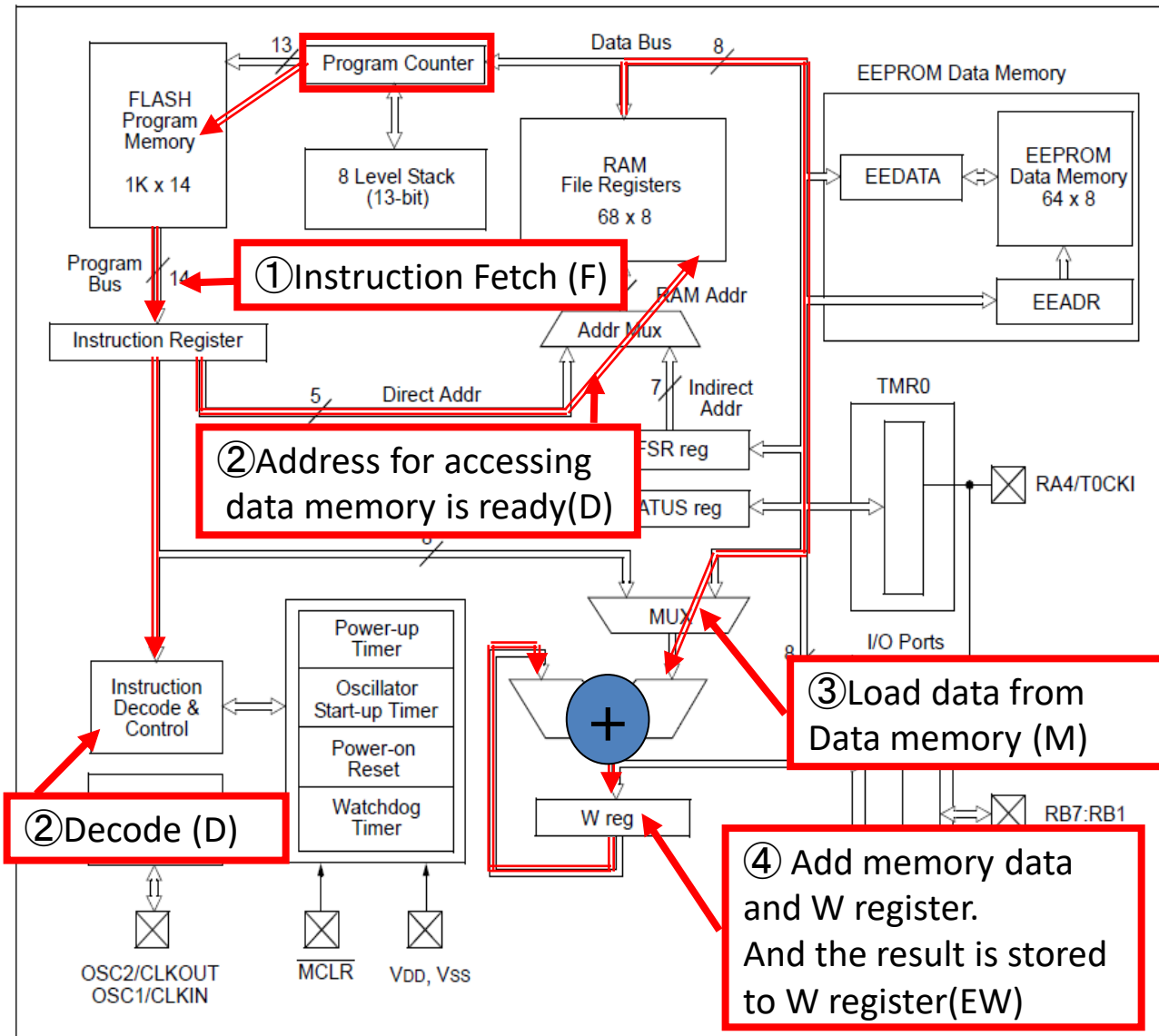
addwf Address, w

① Instruction pointed out by program counter is stored to instruction register.

② Instruction is decoded. And address for accessing data memory is ready.

③④ Add memory data and W register. And the result is stored to W register.

FIGURE 1-1: PIC16F84A BLOCK DIAGRAM



Instruction Set of PIC16F84A (1)

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2

x: Don't care

C: Carry flag, DC: Digit Carry flag

Z: Zero flag

 : Explained instructions in previous

f: Data memory Address
d: Data memory when d=1 (f)
W register when d=0 (W)

Instruction Set of PIC16F84A (2)

BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}, \overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}, \overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

f: Address of Data memory
b: Bit index for bit operation
k: Constant value (literal)

x: Don't care

C: Carry flag, DC: Digit Carry flag

Z: Zero flag

TO, PD: Status for internal states of processor

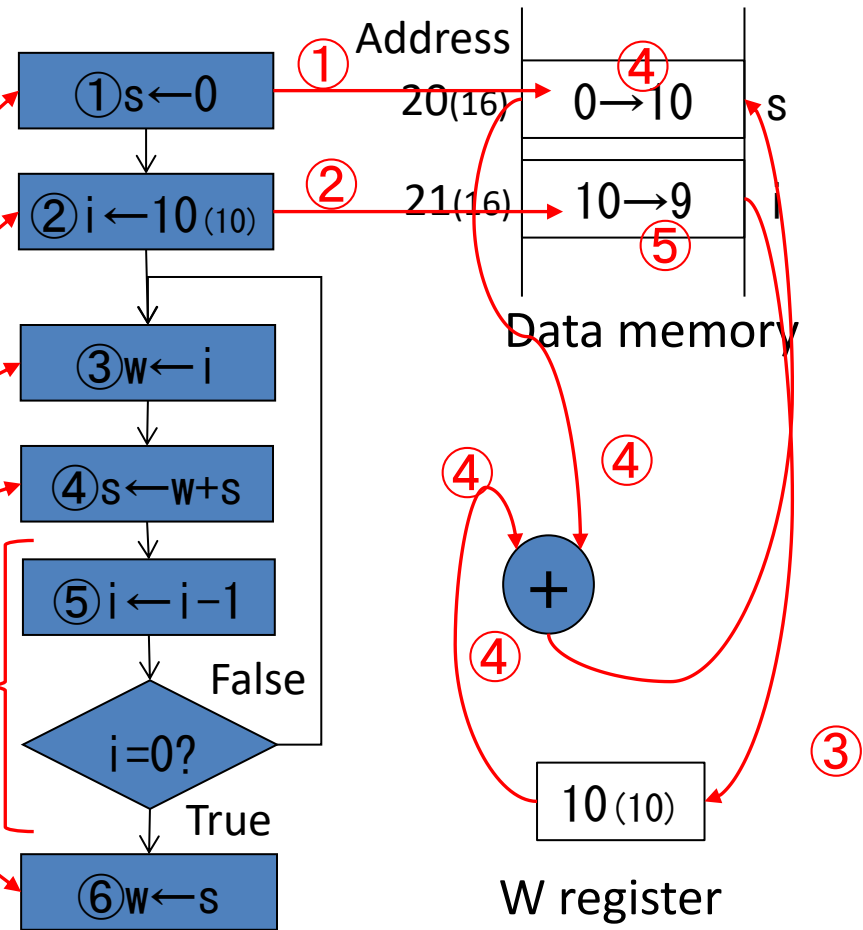
 : Explained instruction in previous

Summation program from 1 to 10

$$s = \sum_{i=1}^{10} i = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55$$

```

INCLUDE "p16f84a.inc"
LIST      P=PIC16F84A
ORG       0
clrf      H' 20'      ; s ← 0
movlw     D' 10'      ; w ← 10
movwf     H' 21'      ; i ← 10
LOOP      movf      H' 21', w    ; w ← i
          addwf     H' 20', f    ; s ← i + s
          decfsz    H' 21', f    ; i ← i - 1
          goto      LOOP
          movf      H' 20', w    ; w ← s
          clrw
          end
    
```



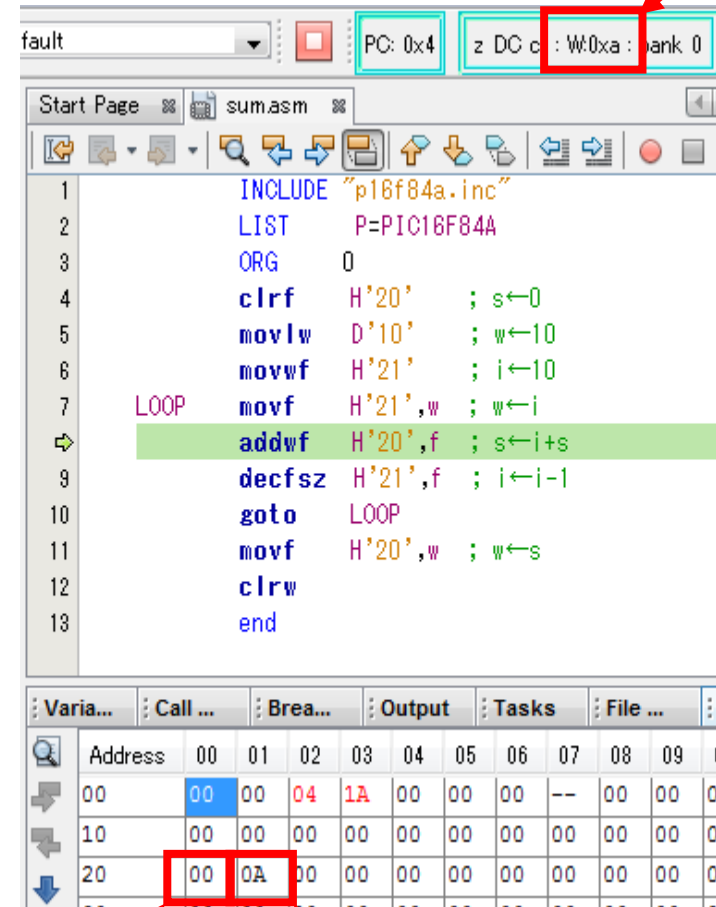
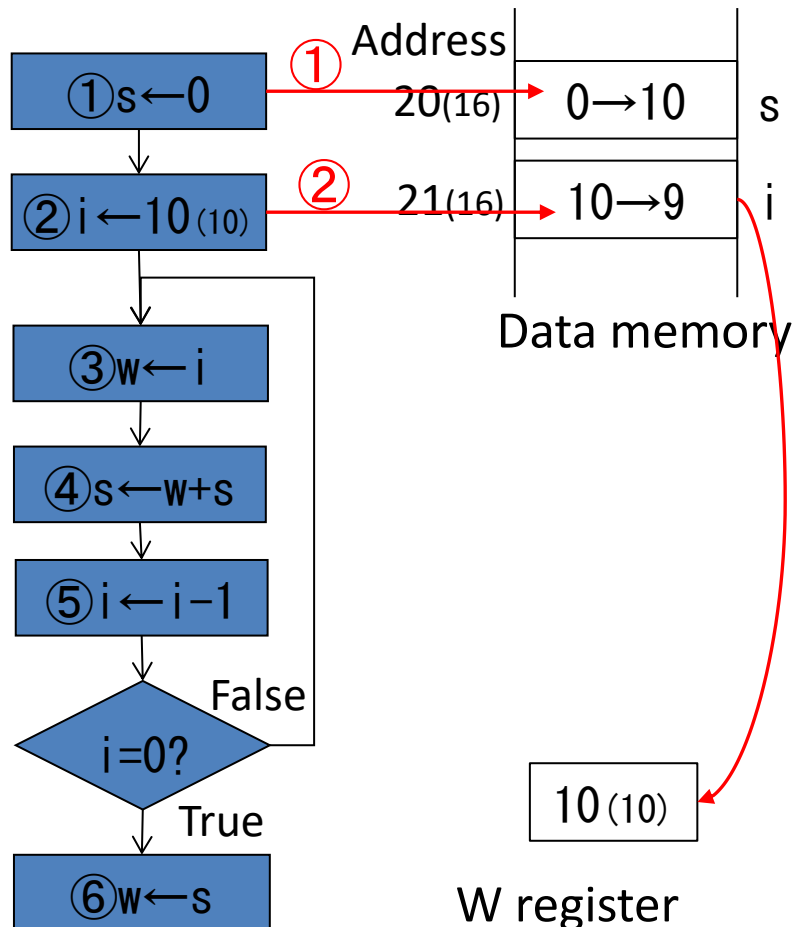
Flow chart

When $i-1$ equals 0, next instruction is canceled and after the next instruction will be executed.

Simulation Result (1)

- Simulator: MPLAB X IDE by Microchip
 - The first iteration after executed ①②③ instructions

W=10₍₁₀₎



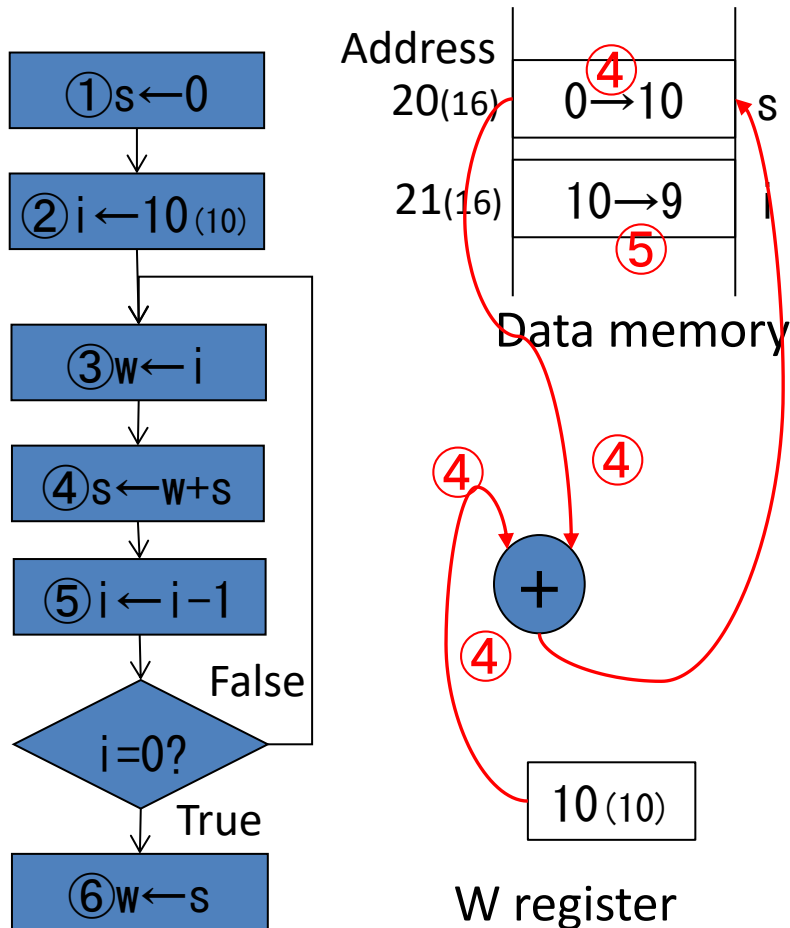
s=0₍₁₀₎

i=10₍₁₀₎

Simulation Result (2)

- Simulator: MPLAB X IDE by Microchip
 - The first iteration after executed ④⑤ instructions

W=10(10)



Assembly Code:

```

1  INCLUDE "p18f84a.inc"
2  LIST P=PIC18F84A
3  ORG 0
4  clrf H'20' ; s ← 0
5  movlw D'10' ; w ← 10
6  movwf H'21' ; i ← 10
7  LOOP movf H'21',w ; w ← i
8      addwf H'20',f ; s ← i + s
9      decfsz H'21',f ; i ← i - 1
10     goto LOOP
11     movf H'20',w ; w ← s
12     clrw
13     end
    
```

Memory Dump:

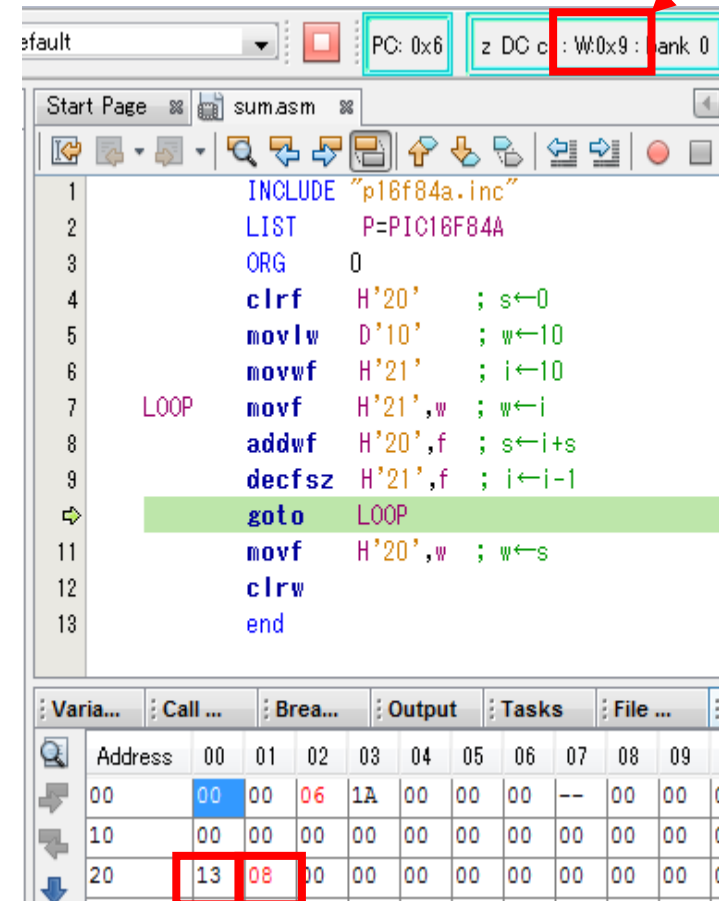
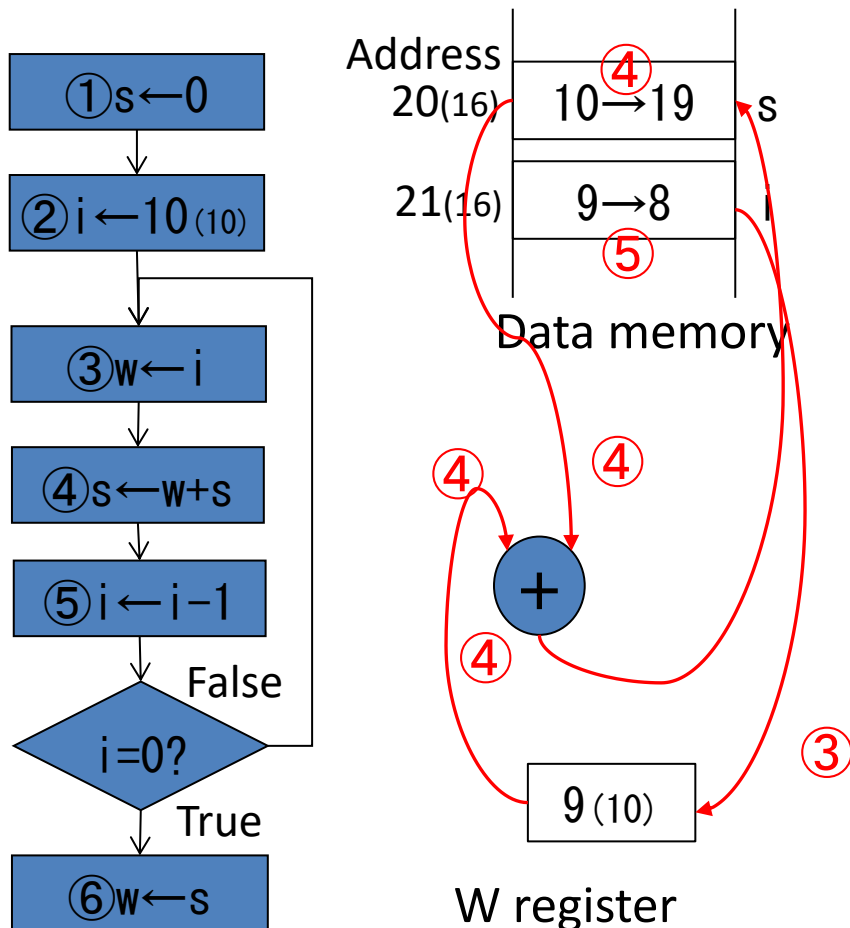
Address	00	01	02	03	04	05	06	07	08	09	0A	0B
00	00	00	06	18	00	00	00	--	00	00	00	00
10	00	00	00	00	00	00	00	00	00	00	00	00
20	0A	09	00	00	00	00	00	00	00	00	00	00

s=10(10)

i=9(10)

Simulation Result (3)

- Simulator: MPLAB X IDE by Microchip
 - The second iteration after executed ③④⑤ instructions

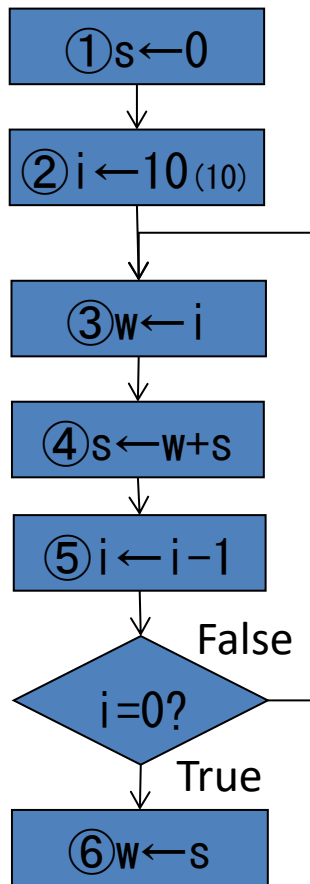


s=19(10)

i=8(10)

Simulation Result (4)

- Simulator: MPLAB X IDE by Microchip
 - The tenth iteration after executed ④⑤⑥ instructions



Flow chart

Address		
20(16)	54 → 55	s
21(16)	1 → 0	i

Data memory

W register

55(10)

The screenshot shows the MPLAB X IDE interface. The assembly code window displays the following code:

```

1  INCLUDE "p16f84a.inc"
2  LIST P=PIC16F84A
3  ORG 0
4  clrf H'20' ; s ← 0
5  movlw D'10' ; w ← 10
6  movwf H'21' ; i ← 10
7  LOOP movf H'21',w ; w ← i
8      addwf H'20',f ; s ← i+s
9      decfsz H'21',f ; i ← i-1
10     goto LOOP
11     movf H'20',w ; w ← s
12     clrw
13     end
    
```

The register window at the bottom shows the state of the registers. The W register (00000000) contains the value 37 (0x25), which is highlighted with a red box. The PC register (00000000) contains the value 00 (0x00), which is also highlighted with a red box. The status bar at the bottom shows the current address as 00000000.

s=55(10)

i=0(10)

W=55(10)

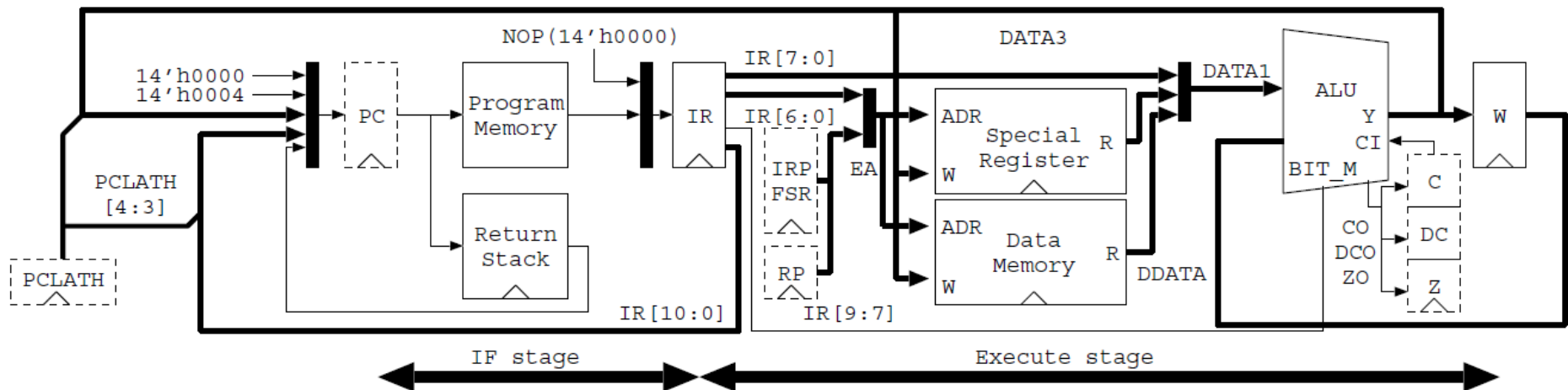
Differences from the original

	Original PIC16	Our design
Pipeline	4 cycles / 1 stage	1 cycle / 1 stage Different I/O timing
Clock	Selectable clock, external or internal OSC	External clock only
Sleep mode	Low power mode	Repeat NOP instruction
Watch Dog timer (WDT)	Available	None
Timer (TMR0)	Available	None
Prescaler	Available	None
Interrupt	Available	None
Flash mem.	Available	None

Block Diagram

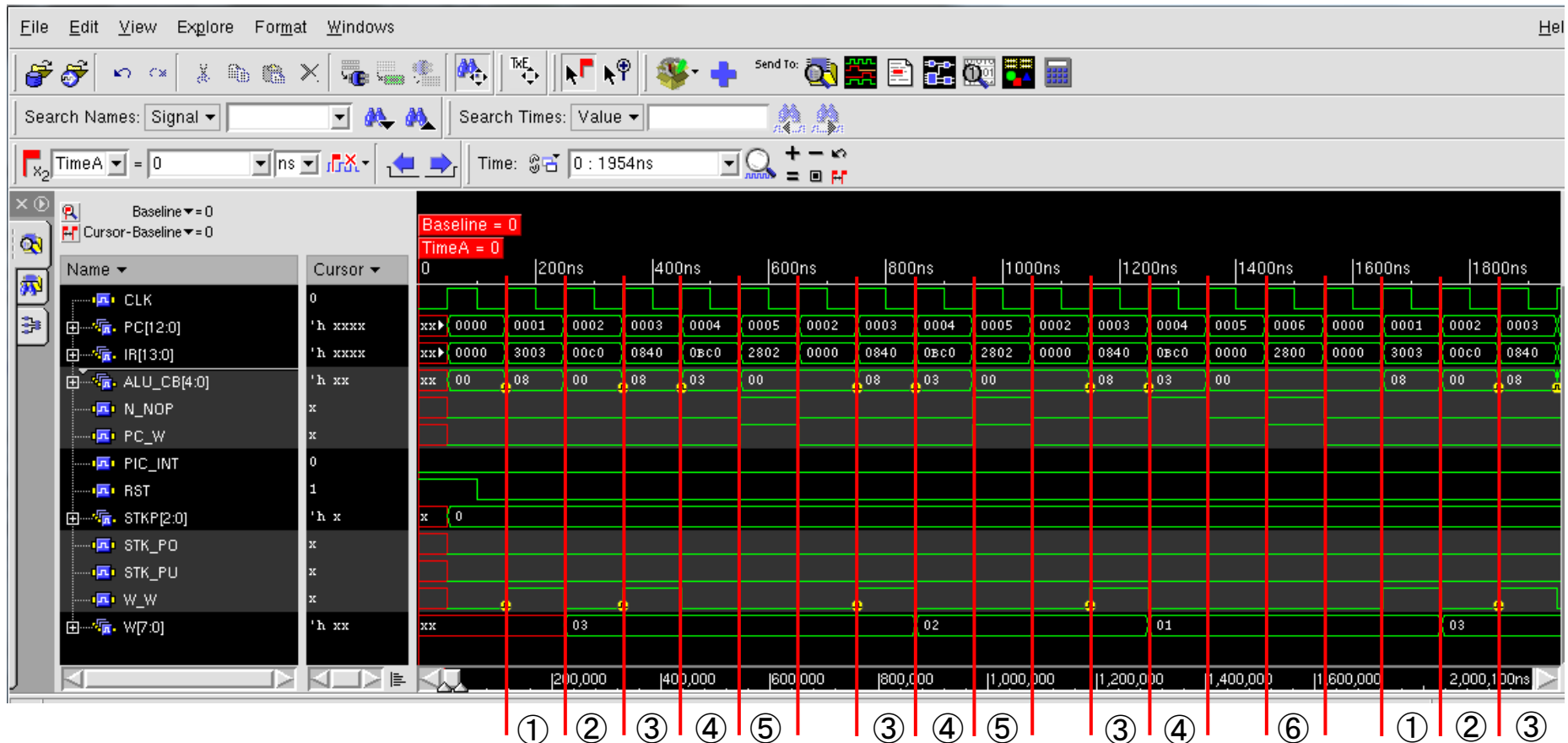
- 2-stage Pipeline
 - Instruction Fetch (IF) stage
 - Execution (E) stage

Each stage is executed in parallel
- Critical Path
 - IR → Data Memory → ALU → Data Memory
 - Operating frequency cannot be faster because of Read/Write accessing of Data Memory in one cycle.

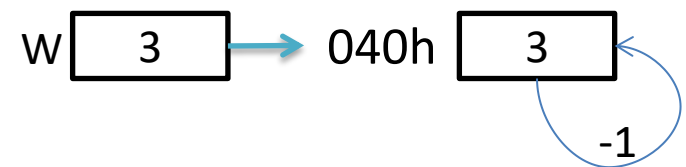


Behavior of 2 Stage Pipeline Processor

Behavior of PIC16 compatible processor designed in our Lab. (Note that original PIC 16 has 4 cycles in a one stage)



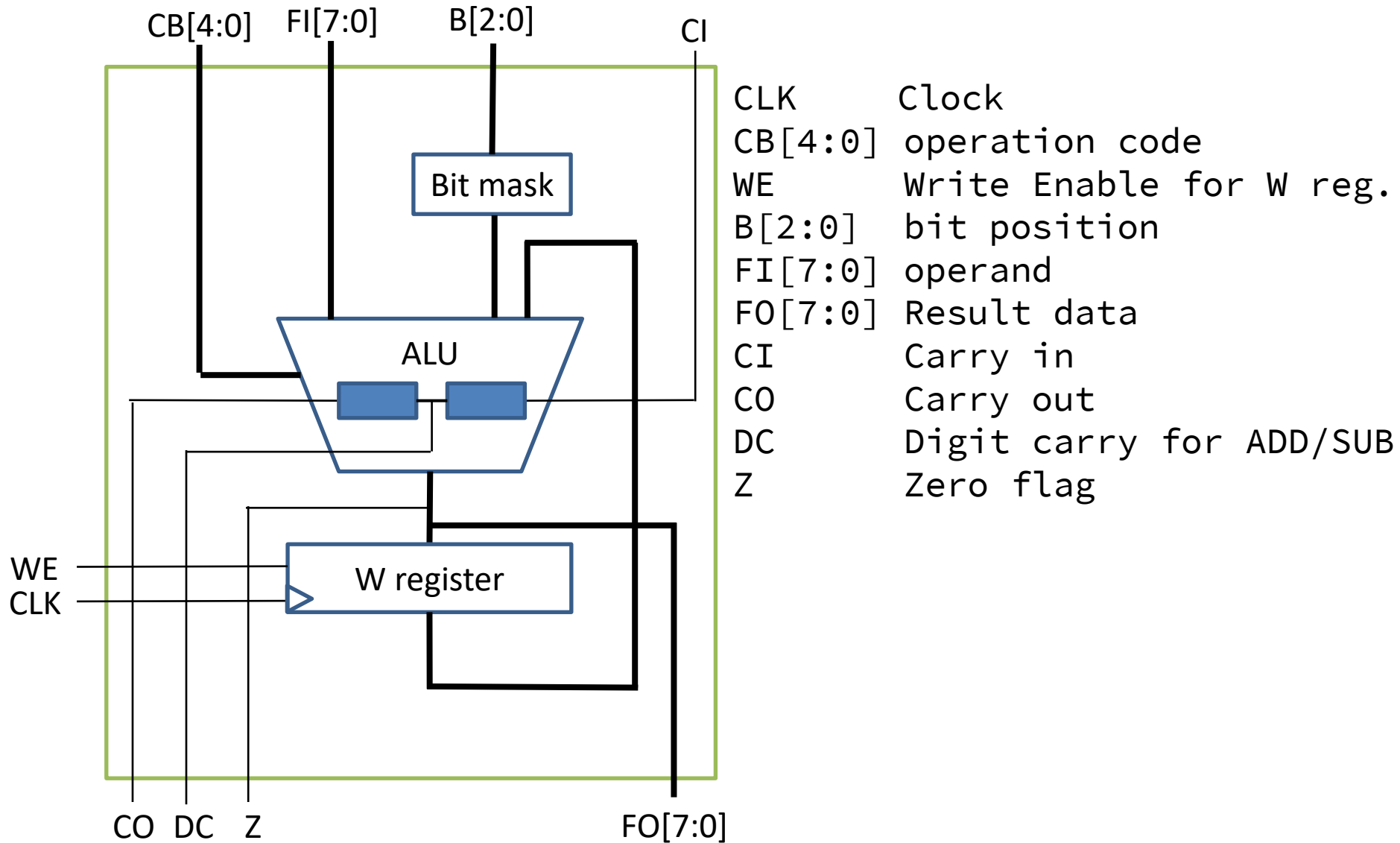
0000	0001	ORG 0
0000 3003	00002 L2:	① MOVLW 3
0001 00C0	00003	② MOVWF 040h
0002 0840	00004 L1:	③ MOVF 040h, 0
0003 0BC0	00005	④ DECFSZ 040h, 1
0004 2802	00006	⑤ GOTC L1
0005 2800	00007	⑥ GOTC L2
	00008	end



ALU Design

DESIGN EXAMPLE

ALU Design for PIC16



alu.v : module alu

Refer the skeleton code, “alu.v”

```
//  
// ALU for PIC16  
//  
`include "alu_op.v"  
  
module alu ( CLK, CB, WE, B, FI, FO, CI, CO, DC, Z );  
    input          CLK; // Clock  
    input  [4:0]   CB;  // operation code  
    input          WE;  // Write enable for W register  
    input  [2:0]   B;   // bit position  
    input  [7:0]   FI;  // left operand  
    output [7:0]   FO;  // result data  
    input          CI;  // Carry in  
    output         CO;  // Carry out (ADD:Carry/SUB:Borrow)  
    output         DC;  // Digit Carry for ADD/SUB(Half carry)  
    output         Z;   // Zero  
  
    ...  
  
endmodule
```

alu_op.v : ALU operation code

```
// Control code for PIC16 ALU
`define      IPSW      5'b00000 // Pass W
`define      ICLR      5'b00001 // Clear F and W
`define      ISUB      5'b00010 // Arithmetic Subtract
`define      IDEC1     5'b00011 // Decrement for DECF
`define      IOR       5'b00100 // Logical OR
`define      IAND      5'b00101 // Logical AND
`define      IXOR      5'b00110 // Logical exclusive OR
`define      IADD      5'b00111 // Arithmetic Add
`define      IPSF      5'b01000 // Pass F
`define      INTF      5'b01001 // Logical Complement F (NOT)
`define      IINC1     5'b01010 // Increment for INCF
`define      IDEC2     5'b01011 // Decrement for DECFSZ
`define      IRRF      5'b01100 // Rotate Right F with carry
`define      IRLF      5'b01101 // Rotate Left F with carry
`define      ISWP      5'b01110 // Nibble swap F
`define      IINC2     5'b01111 // Increment for INCFSZ
`define      IBCF      5'b100?? // Bit Clear F
`define      IBSF      5'b101?? // Bit Set F
`define      IBTF      5'b11??? // Bit Test F
```

Instruction Set of PIC16F84A (1)

Mnemonic, Operands		Description	Cycles	14-Bit Opcode				Status Affected	Notes
				MSb		LSb			
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2

f: Address field of Data memory
d: Data memory when d=0 (f)
W register when d=1 (W)

x: Don't care

C: Carry flag, DC: Digit Carry flag

Z: Zero flag

 : ALU operation code

Instruction Set of PIC16F84A (2)

BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	$\overline{TO}, \overline{PD}$	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	$\overline{TO}, \overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

f: Address field of Data memory

b: Bit index for bit operation

k: Constant value (literal)

x: Don't care

C: Carry flag, DC: Digit Carry flag

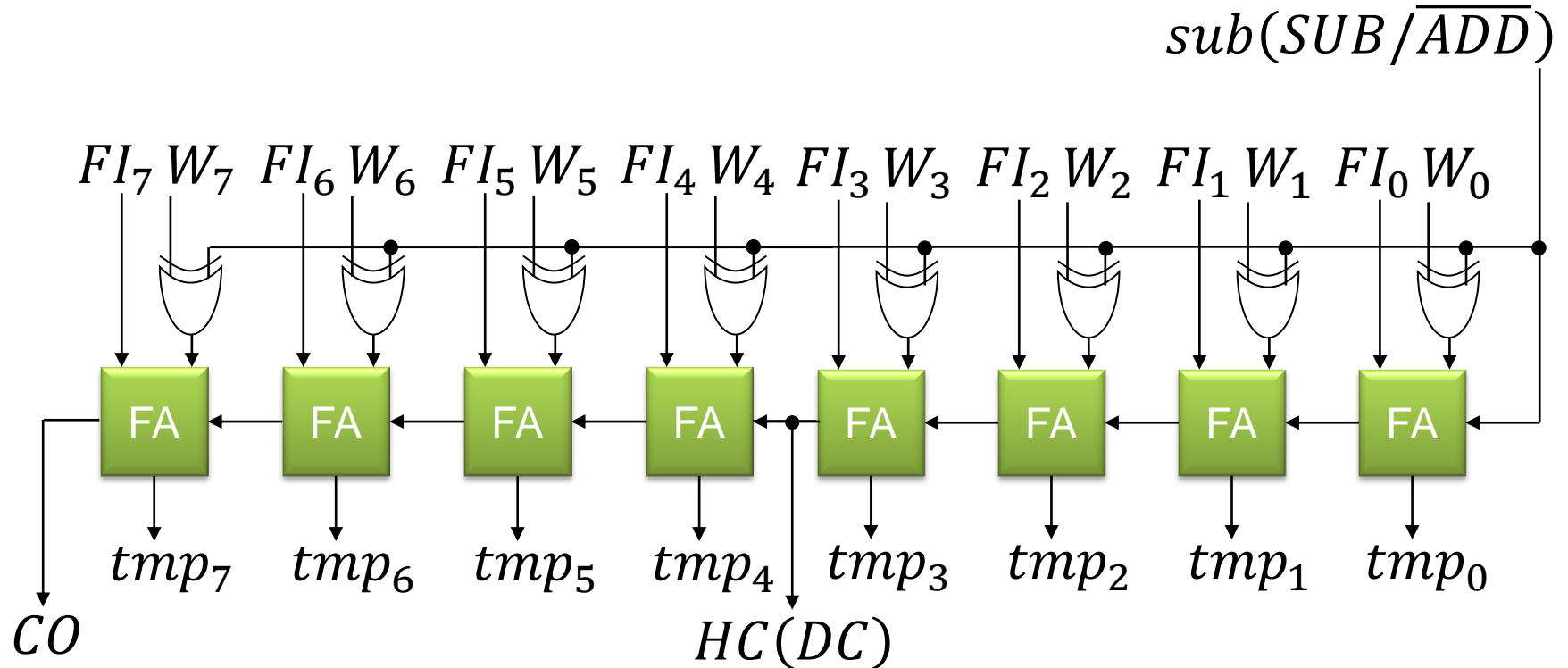
Z: Zero flag

TO, PD: Status for internal states of processor

 : ALU operation code

 : ALU codes for LW type inst. are controlled by decoder

ADD/SUB unit for PIC ALU



Carry out
(ADD: Carry/SUB: $\overline{\text{Borrow}}$)

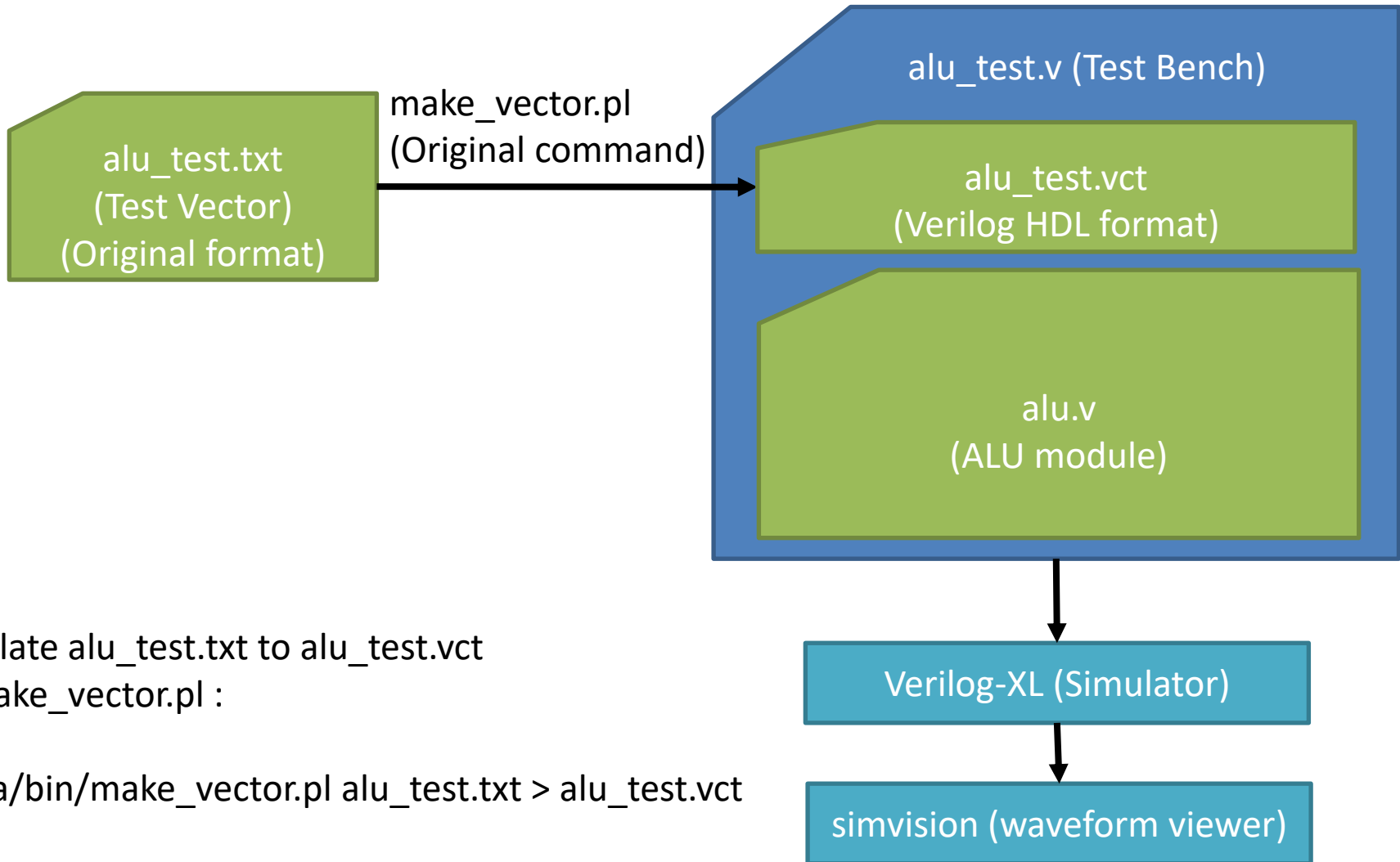
Digit Carry
(ADD: Carry/SUB: $\overline{\text{Borrow}}$)

```
`IADD, `ISUB:
begin
    { DC, tmp[3:0] } = { 1'b0, ... } + { 1'b0, (...) ? ..., ... } + sub;
    tmp[8:4]        = { 1'b0, ... } + { 1'b0, (...) ? ..., ... } + DC;
```


ALU Simulation

DESIGN EXAMPLE

ALU Simulation



alu_test.v : Test Bench

```
`timescale 1ns/1ns
`include "alu_op.v"

module alu_test;
    reg    [7:0]  FI;
    reg    [2:0]  B;
    reg    [4:0]  CB;
    reg          CLK, WE, CO;

    wire    [7:0]  FO;
    wire          DC, CI, Z;

    initial
        begin
            $shm_open("waves.shm");
            $shm_probe("as");
        end

    `include "alu_test.vct"

    alu alu1 ( .CLK(CLK), .CB(CB), .WE(WE), .B(B),
               .FI(FI), .FO(FO), .CI(CO), .CO(CI), .DC(DC), .Z(Z) );

    always @( posedge CLK )
        CO <= CI;

endmodule
```

} Time accuracy for simulation (unit time 1ns/precision 1ns)

} Input signals (reg type variable)

} Output signals (wire type variable)

} Directives for signal monitoring
by verilog-XL and simvision, Cadence EDA tools

} Test vector

} Carry flag register

alu_test.txt : test vector

```
# input
CB[4:0]
B[2:0]
FI[7:0]
WE
# clock
CLK 20 # 20ns (50MHz)
# testvector
# CB[4:0] B[2:0] FI[7:0] WE
10 `IPSF 0 8'hFF 0 # Pass F
20 `IPSF 0 8'h55 0 # Pass F
20 `IPSF 0 8'hAA 0 # Pass F
20 `ICLR 0 8'hAA 0 # Clear F
20 `IINC1 0 8'h00 1 # Increment F, W=1
20 `ISUB 0 8'h01 0 # Subtract 1 from F=1
20 `IDEC1 0 8'h01 1 # Decrement F, W=0
20 `IPSF 0 8'h55 1 # Pass F, W=0x55
20 `IAND 0 8'h55 0 # 0x55 & 0x55
20 `IAND 0 8'hAA 0 # 0xAA & 0x55
20 `IOR 0 8'h55 0 # 0x55 | 0x55
20 `IOR 0 8'hAA 0 # 0xAA | 0x55
20 `IXOR 0 8'h55 0 # 0x55 ^ 0x55
20 `IXOR 0 8'hAA 0 # 0xAA ^ 0x55
20 `IPSW 0 8'h00 0 # Pass W, FO<=W
20 `IRRF 0 8'h55 1 # Rotate Right 0x55, W<= result
20 `IRLF 0 8'hAA 0 # Rotate Left 0xAA
20 `IBSF 2 8'hAA 0 # Bit Set 2
20 `IBCF 1 8'hAA 0 # Bit Clear 1
20 `IBTF 4 8'hAA 0 # Bit Test 4, Check Z flag
20 `IBTF 5 8'hAA 0 # Bit Test 5, Check Z flag
```

Definition of input signals
"# input" is start line of this definition, not comment

Definition of clock signal
"# clock" is start line of this definition, not comment

Definition of test vector
"#testvector" is start line of this def.

One line is one vector.
Column parameters are

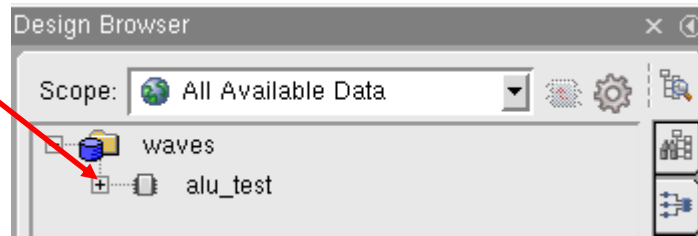
- Delay
- CB
- B
- FI
- WE, and
- # comment.

Verilog simulation

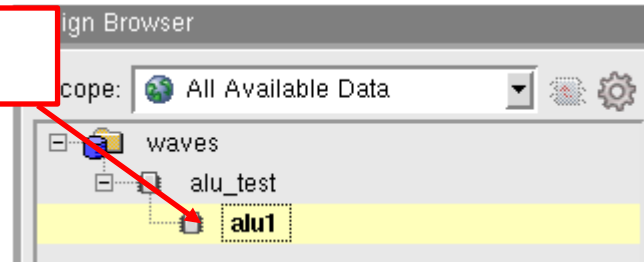
- Simulator and waveform viewer are executed on “calc1.st.cs.kumamoto-u.ac.jp”
- For set up the environment, type follows;
ssh -X calc1.st.cs.kumamoto-u.ac.jp
source ~kuga/setup/creative2016
- For the simulation, type follows;
verilog alu_test.v alu.v
simvision waves.shm/waves.trn

Waveform viewer: simvision

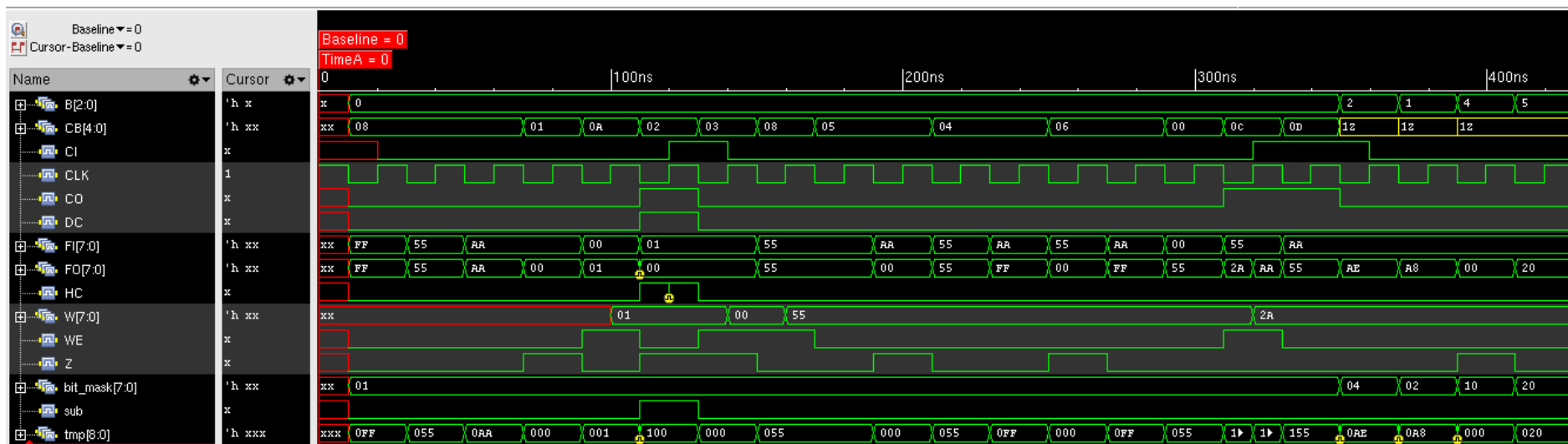
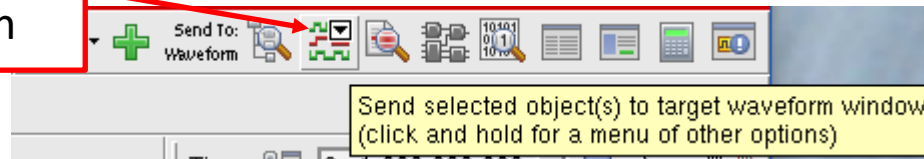
① click “+”



② select “alu1”



② push this icon



PIC16 core design

DESIGN EXAMPLE

Differences from the original

	Original PIC16	Our design
Pipeline	4 cycles / 1 stage	1 cycle / 1 stage Different I/O timing
Clock	Selectable clock, external or internal OSC	External clock only
Sleep mode	Low power mode	Repeat NOP instruction
Watch Dog timer (WDT)	Available	None
Timer (TMR0)	Available	None
Prescaler	Available	None
Interrupt	Available	None
Flash mem.	Available	None

Instruction Set of PIC16F84A (1)

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2

x: Don't care

C: Carry flag, DC: Digit Carry flag

Z: Zero flag

: ALU operation code

f: Address field of Data memory
d: Data memory when d=1 (f)
W register when d=0 (W)

Instruction Set of PIC16F84A (2)

BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDT	-	Clear Watchdog Timer	1	00	0000	0110	0100	TO,PD	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	TO,PD	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

f: Address field of Data memory

b: Bit index for bit operation

k: Constant value (literal)

x: Don't care

C: Carry flag, DC: Digit Carry flag

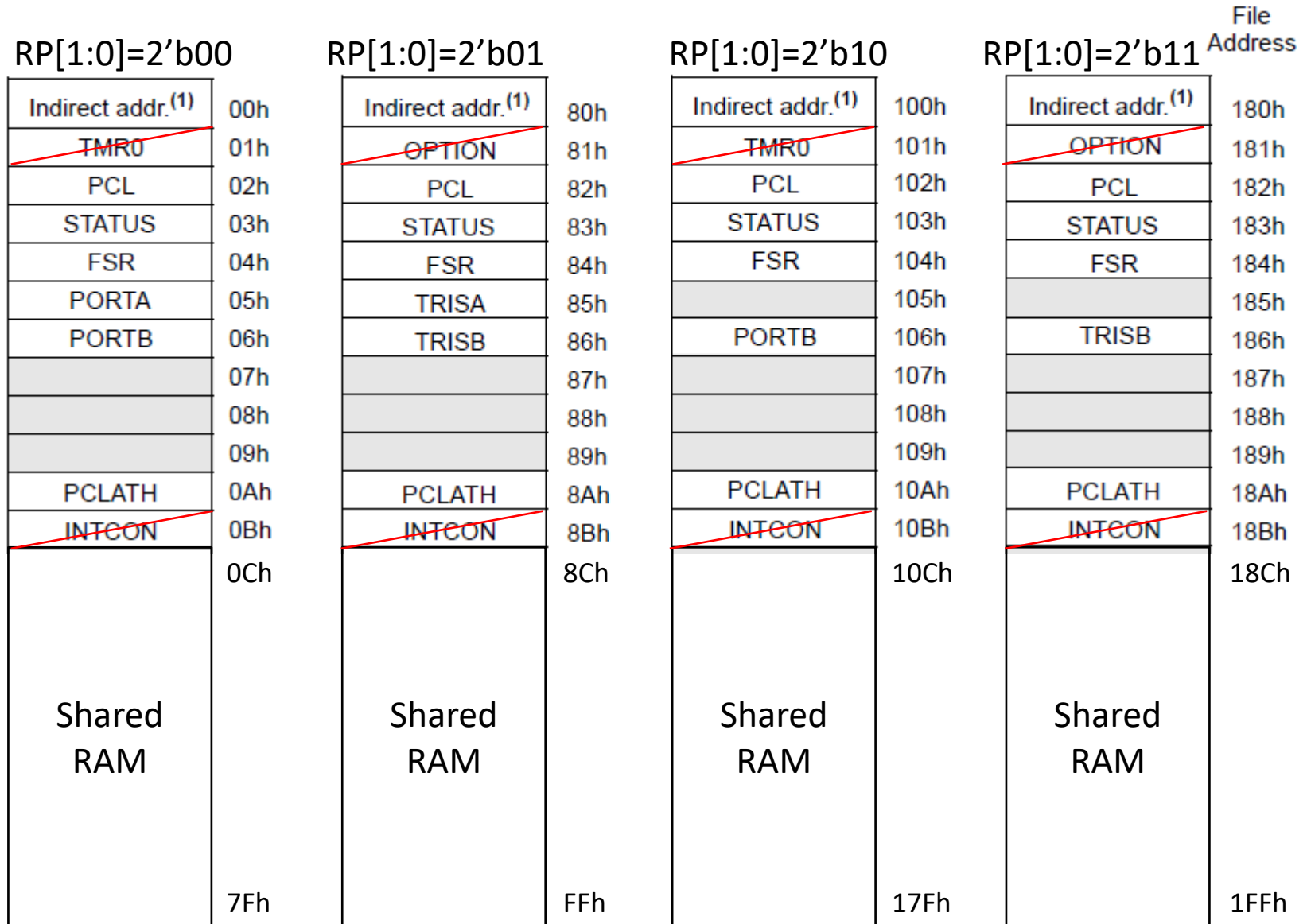
Z: Zero flag

TO, PD: Status for internal states of processor

 : Unimplemented

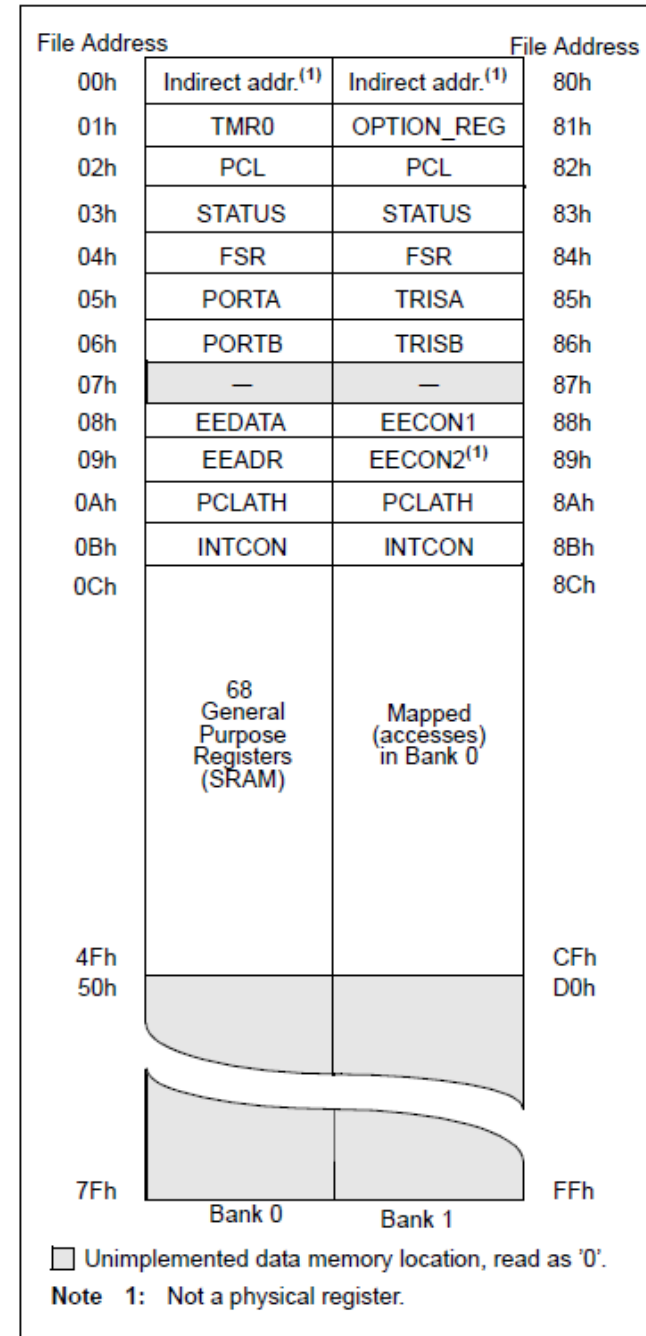
 : Restricted implementation (cannot wake-up from sleep mode)

Data memory map in our design



Data memory map in PIC16F84A

**FIGURE 2-2: REGISTER FILE MAP -
PIC16F84A**



Data memory map in PIC16F648A

FIGURE 4-3: DATA MEMORY MAP OF THE PIC16F648A

								File Address
Indirect addr. ⁽¹⁾	00h	Indirect addr. ⁽¹⁾	80h	Indirect addr. ⁽¹⁾	100h	Indirect addr. ⁽¹⁾	180h	
TMR0	01h	OPTION	81h	TMR0	101h	OPTION	181h	
PCL	02h	PCL	82h	PCL	102h	PCL	182h	
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h	
FSR	04h	FSR	84h	FSR	104h	FSR	184h	
PORTA	05h	TRISA	85h		105h		185h	
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h	
	07h		87h		107h		187h	
	08h		88h		108h		188h	
	09h		89h		109h		189h	
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah	
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh	
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch	
	0Dh		8Dh		10Dh		18Dh	
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh	
TMR1H	0Fh		8Fh		10Fh		18Fh	
T1CON	10h		90h					
TMR2	11h		91h					
T2CON	12h	PR2	92h					
	13h		93h					
	14h		94h					
CCPR1L	15h		95h					
CCPR1H	16h		96h					
CCP1CON	17h		97h					
RCSTA	18h	TXSTA	98h					
TXREG	19h	SPBRG	99h					
RCREG	1Ah	EEDATA	9Ah					
	1Bh	EEADR	9Bh					
	1Ch	EECON1	9Ch					
	1Dh	EECON2 ⁽¹⁾	9Dh					
	1Eh		9Eh					
CMCON	1Fh	VRCON	9Fh		11Fh			
	20h		A0h		120h			
General Purpose Register 80 Bytes		General Purpose Register 80 Bytes		General Purpose Register 80 Bytes				
	6Fh		EFh		16Fh			
16 Bytes	70h	accesses 70h-7Fh	F0h	accesses 70h-7Fh	170h			
	7Fh		FFh		17Fh			
Bank 0		Bank 1		Bank 2		Bank 3		

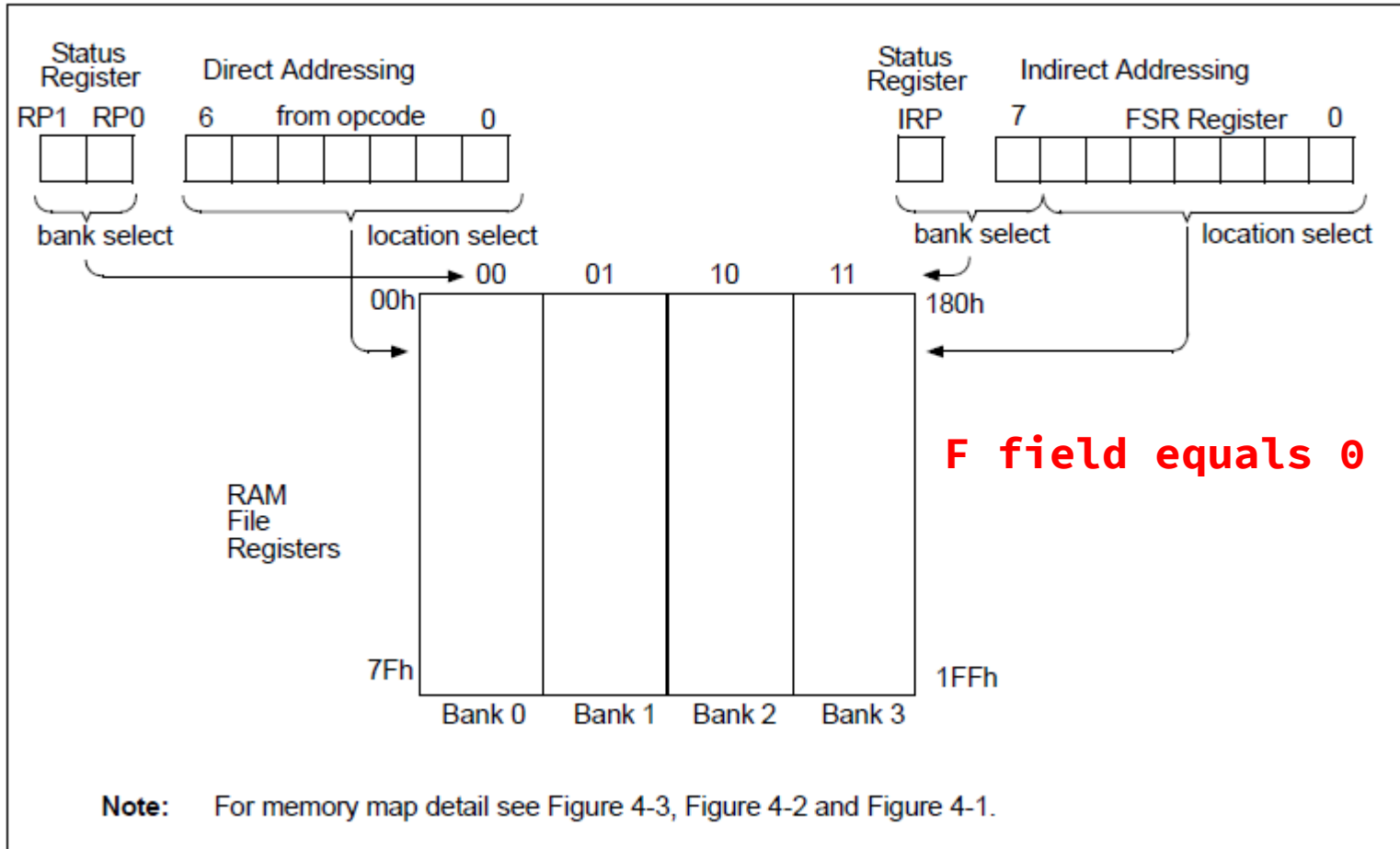
■ Unimplemented data memory locations, read as '0'.

Note 1: Not a physical register.

Effective Address for Data Memory

```
// Effective Address
wire [  :  ] EA;
assign EA = ( `IRF ==      ) ? {      ,      } : {      ,      };
```

FIGURE 4-5: DIRECT/INDIRECT ADDRESSING PIC16F627A/628A/648A



Program Counter and Return Stack

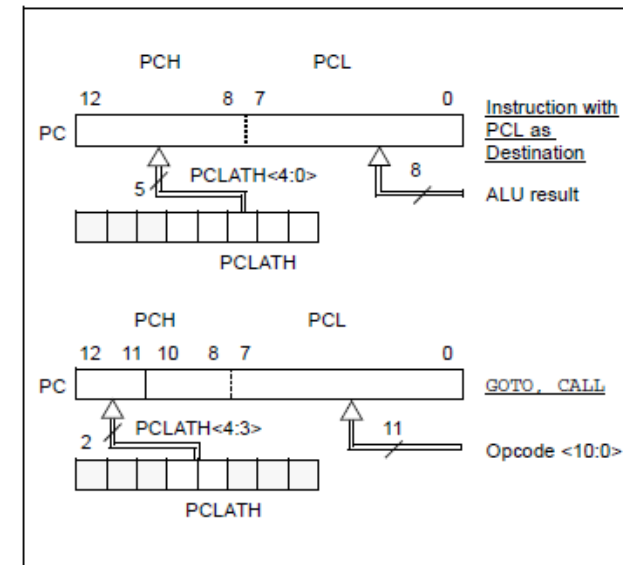
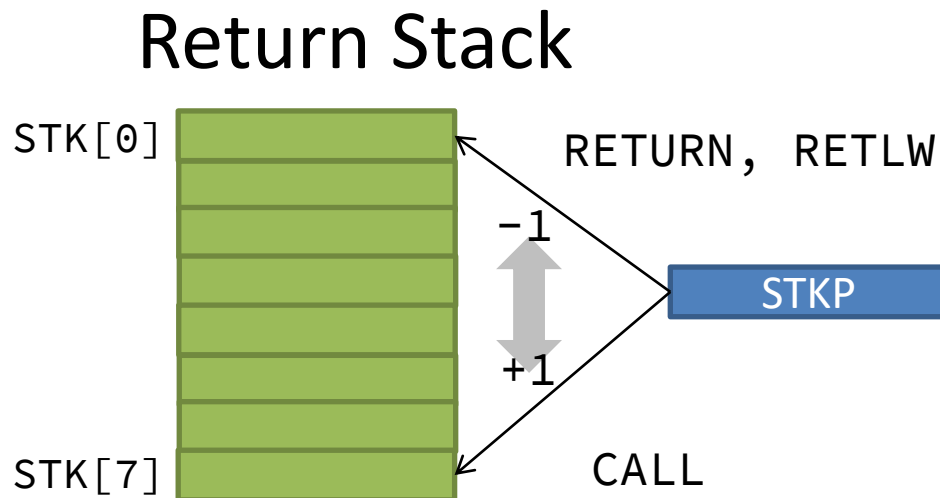
```
// Program Counter
```

```
always @( posedge CLK )
    if(                ) PC <=                ; else // RESET
    if(                ) PC <= {                ,                }; else // CALL, GOTO
    if(                ) PC <= {                ,                }; else // write PCL register
    if(                ) PC <= STK[                ]; else // RETURN, RETLW
    if(                ) PC <=                ; else // SLEEP mode
        PC <= PC + 1;
```

```
// Return Stack
```

```
always @( posedge CLK )
    begin
        if(                ) STKP<=                ;
        if(                ) begin STK[                ]<=                ; STKP<=                ; end else // for CALL
        if(                ) STKP<=                ; // for RETxx
    end
```

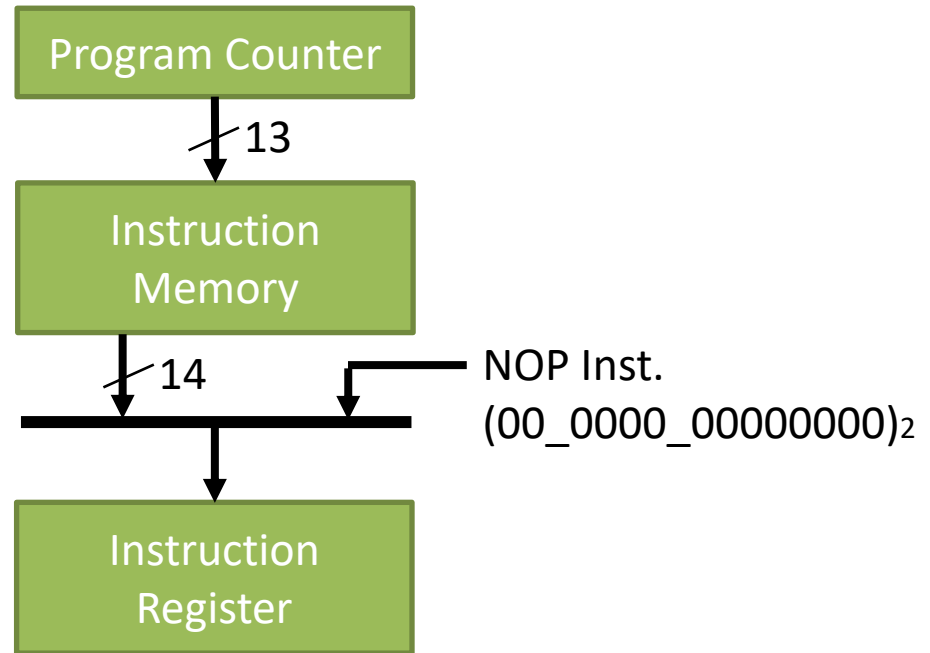
FIGURE 4-4: LOADING OF PC IN DIFFERENT SITUATIONS



Instruction Memory and Register

```
//  
// Instruction Memory (8k word)  
//  
reg [ : ] IMEM[ : ];  
initial  
begin  
    $readmemh( PROG, IMEM );  
end
```

```
//  
// Instruction Register  
//  
always @( posedge CLK )  
    if( || || )  
        IR <= ; else // if CALL, RET, cond.SKIP  
        IR <= ; // Instruction fetch
```



Next inst. is changed to NOP when PC is modified

by GOTO, CALL, RETURN, RETLW, RETFIE, INCFZS, DECFZS, BTFSC, BTFSS, SLEEP inst's SLEEP mode and PCL register.

Decoder and Control Signal Generation

ALU_CB	Operation code for ALU
F_W	Write timing for Special register or Data Memory
W_W	Write timing for W register
Z_W	Write timing for Z flag
DC_W	Write timing for Digit Carry flag
C_W	Write timing for Carry flag
nTO_S	Set timing for nTO register
nTO_C	Clear timing for nTO register
nPD_S	Set timing for nPD register
nPD_C	Clear timing for nPD register
STK_PU	Push timing for Return Stack
STK_PO	Pop timing for Return Stack
NOP_S	Change to NOP instruction for next instruction after modified PC
PC_W	Write timing for Program Counter
WDT_C	Clear timing for Watch Dog Timer
SLP_S	Set timing for SLEEP mode

Decoder and Control Signal Generation

```
// Decode & Control
always @( IR or Z0 )
begin
    ALU_CB=IR[  :  ];
    F_W=0; W_W=0; Z_W=0; DC_W=0; C_W=0;
    nTO_S=0; nTO_C=0; nPD_S=0; nPD_C=0;
    STK_PU=0; STK_PO=0; NOP_S=0; PC_W=0;
    WDT_C=0;
    SLP_S=0;
    case( IR[__:__] )
        2'b__:
            begin
                W_W = ____&&____;
                F_W = ____&&____;
                case( IR[__:__] )
                    4'b__:
                        case( IR[ ] )
                            1'b0: case( IR[__:__] )
                                7'b____: begin ____; ____; end // RETURN
                                7'b0000_1001: ; end // RETFIE
                                7'b____: begin ____; ____; ____; ____; ____; end // SLEEP
                                7'b110_0100: WDT_C=1; end // CLRWDT
                                default: ; end // NOP
                            endcase
                        1'b1: ; end // MOVWF f
                    endcase
                4'b____: begin ____; end // CLRW, CLRF
                4'b____: begin ____; ____; ____; end // SUBWF
                4'b____: begin ____; end // DECF
                4'b____: begin ____; end // IORWF
                4'b____: begin ____; end // ANDWF
                4'b____: begin ____; end // XORWF
                4'b0111: begin C_W=1; DC_W=1; Z_W=1; end // ADDWF
            end
        ...
    end
end
```

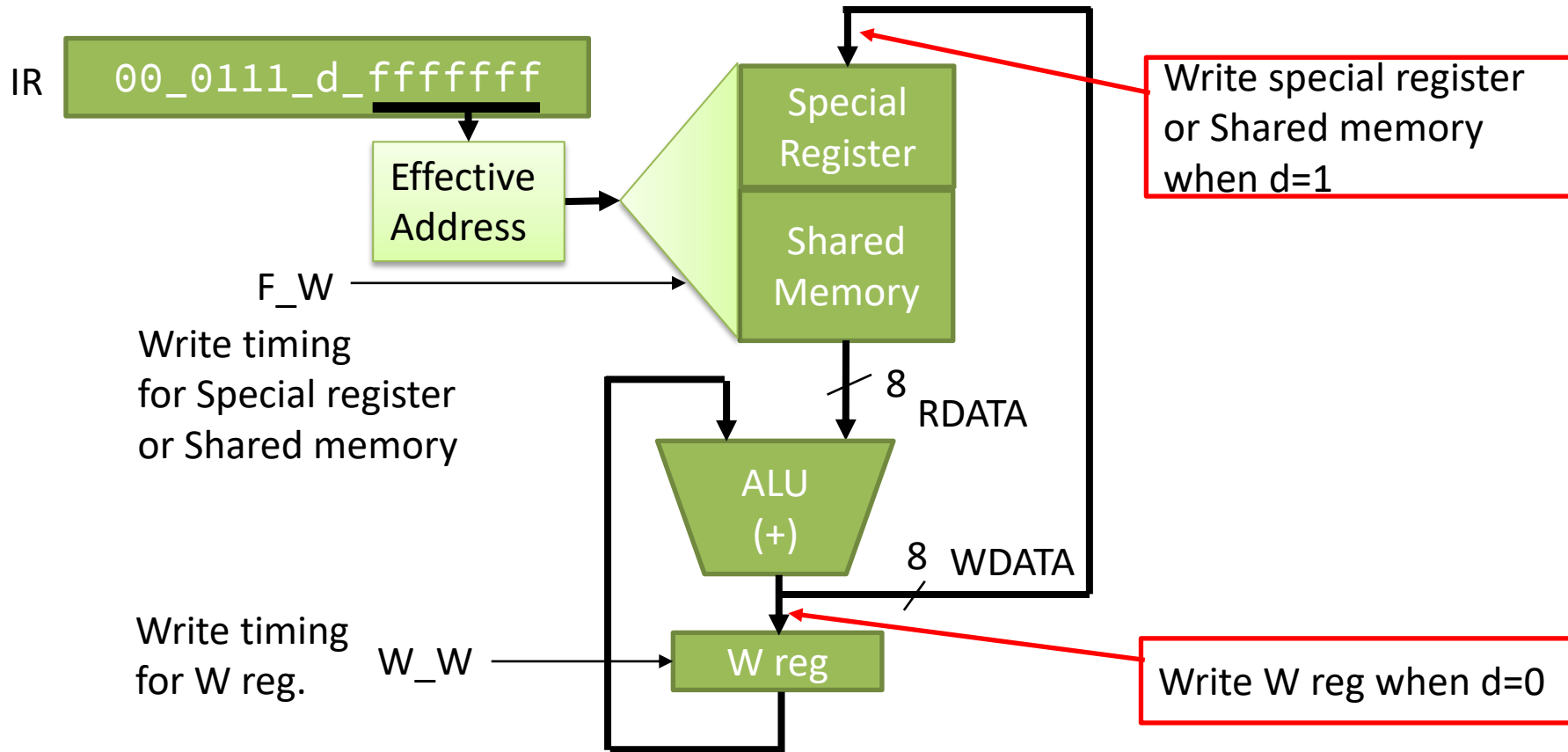
... **TABLE 15-2: PIC16F627A/628A/648A INSTRUCTION SET**

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes
			MSb		LSb			
BYTE-ORIENTED FILE REGISTER OPERATIONS								
ADDWF f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1, 2

Example: ADDWF

TABLE 15-2: PIC16F627A/628A/648A INSTRUCTION SET

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes	
			MSb			LSb			
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1, 2



Special Registers

```
//
// Special Register
// Write
always @( posedge CLK or posedge RST )
begin
    if( RST )
    begin
        C          =      ;
        DC         =      ;
        Z          =      ;
        IRP        =      ;
        RP         =      ;
        nT0        =      ; // nT0=1 on Power-on
        nPD        =      ; // nPD=1 on Power-on
        FSR        =      ;
        PCLATH     =      ;
        PORTA      =      ;
        TRISA      =      ; // All ports for input
        PORTB      =      ;
        TRISB      =      ; // All ports for input
    end
end
```

TABLE 4-3: SPECIAL REGISTERS SUMMARY BANK0

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR Reset ⁽¹⁾	Details on Page
Bank 0											
00h	INDF	Addressing this location uses contents of FSR to address data memory (not a physical register)								xxxx xxxx	28
01h	TMR0	Timer0 Module's Register								xxxx xxxx	45
02h	PCL	Program Counter's (PC) Least Significant Byte								0000 0000	28
03h	STATUS	IRP	RP1	RP0	\overline{TO}	\overline{PD}	Z	DC	C	0001 1xxx	22
04h	FSR	Indirect Data Memory Address Pointer								xxxx xxxx	28
05h	PORTA	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0	xxxx 0000	31
06h	PORTB	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0	xxxx xxxx	36
07h	—	Unimplemented								—	—
08h	—	Unimplemented								—	—
09h	—	Unimplemented								—	—
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of Program Counter					---0 0000	28
0Bh	INTCON	GIE	PEIE	TOIE	INTE	RBIE	TOIF	INTF	RBIF	0000 000x	24
0Ch	PIR1	EEIF	CMIF	RCIF	TXIF	—	CCP1IF	TMR2IF	TMR1IF	0000 -000	26

Special Registers

```

else
begin
  // STATUS register
  if(      ) C    =      ;
  if(      ) DC   =      ;
  if(      ) Z    =      ;
  if(      ) nPD  = 1'b1;
  if(      ) nPD  = 1'b0;
  if(      ) nTO  = 1'b1;
  if(      ) nTO  = 1'b0;
  // Special Register Write
  if(      )
    case(      )
      // 9'b?0_000_0001: TMR0    = WDATA;    // 01    101    Described TMR0 part
      // 9'b?1_000_0001: `OPTION = WDATA;    //      81      181    OPTION not implemented
      // 9'b??_000_0010: PCL     = WDATA;    // 02 82 102 182    Described PC part
      9'b      :              = WDATA;    // 03 83 103 183    STATUS
      9'b      :              = WDATA;    // 04 84 104 184    FSR
      9'b      :              = WDATA;    // 05              PORTA
      9'b      :              = WDATA;    //      85          TRISA
      9'b      :              = WDATA;    // 06      106      PORTB
      9'b      :              = WDATA;    //      86          TIRSB
      // 9'b??_000_0111: ;          // 07 87 107 187          unused
      // 9'b??_000_1000: ;          // 08 88 108 188    EEDATA not implement
      // 9'b??_000_1001: ;          // 09 8A 10A 18A    EEADR  not implement
      9'b??_000_1010:          = WDATA[ : ]; // 0A 8A 10A 18A    PCLATH
      // 9'b??_000_1011: `INTCON = WDATA;    // 0B 8B 10B 18B    INTCON not implement
    endcase
end
end

```

STATUS register

REGISTER 4-1: STATUS – STATUS REGISTER (ADDRESS: 03h, 83h, 103h, 183h)

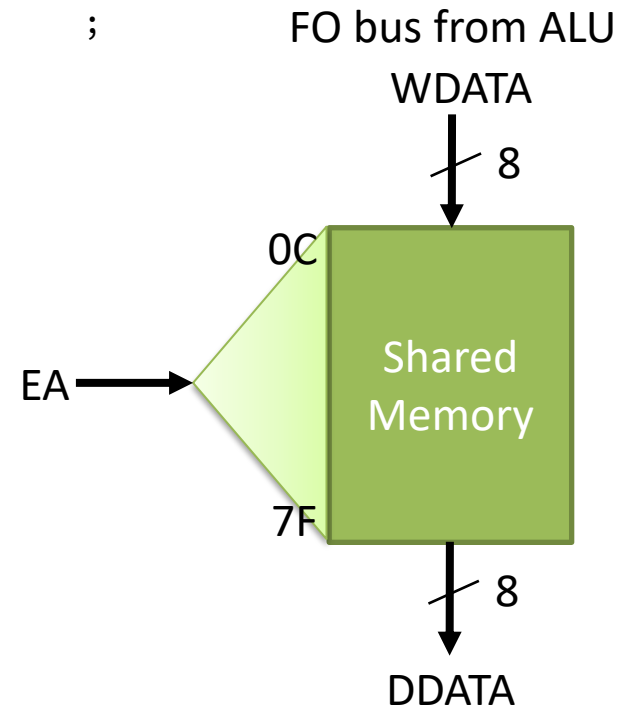
R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C
bit 7							bit 0

Shared Memory

```
// Special Register
...
reg [7:0] RAM[ : ];          // 0C-7F for Shared Memory
...
//
// Data RAM (Write)
//
always @( posedge CLK )
begin
    if(      && (      >=      ) )      <=      ;

end

//
// Data RAM (Read)
//
assign DDATA =      ;
```

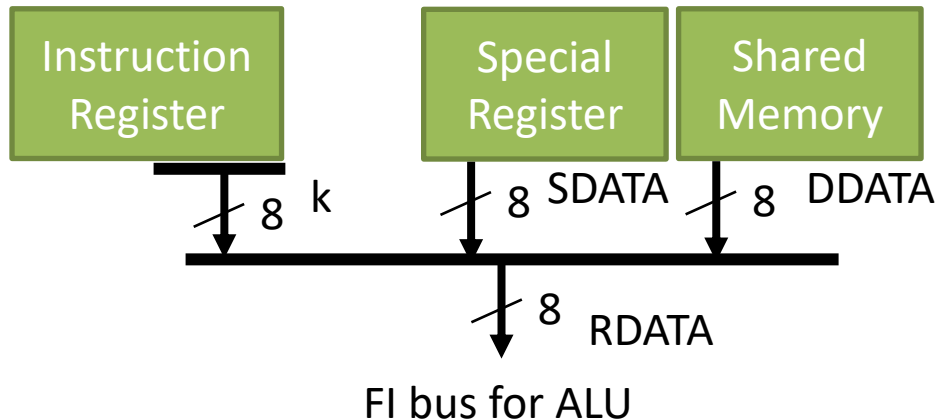


Data path

```

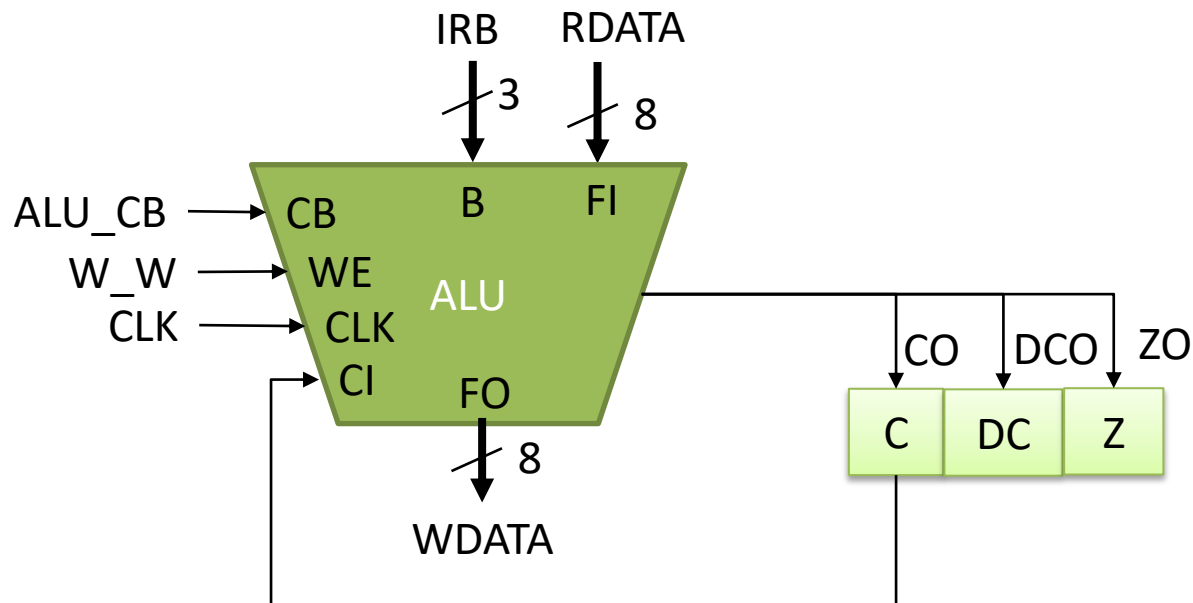
always @( IR or EA or DDATA or SDATA )
begin
    RDATA <= 8'bxxxx_xxxx;           // Default
    if( &IR[ : ] ) RDATA <=          ; else // Literal (k)
        casex( EA )
            9'b          : RDATA <=          ; // PCL (Lower byte of PC)
            9'b          : RDATA <=          ; // STATUS
            9'b          : RDATA <=          ; // FSR
            9'b          : RDATA <=          ; // PORTA, TRISA
            9'b          : RDATA <=          ; // PORTB, TRISB
            9'b          : RDATA <=          ; // PCLATH
            default :      RDATA <=          ; // Shared memory
        endcase
end
end

```



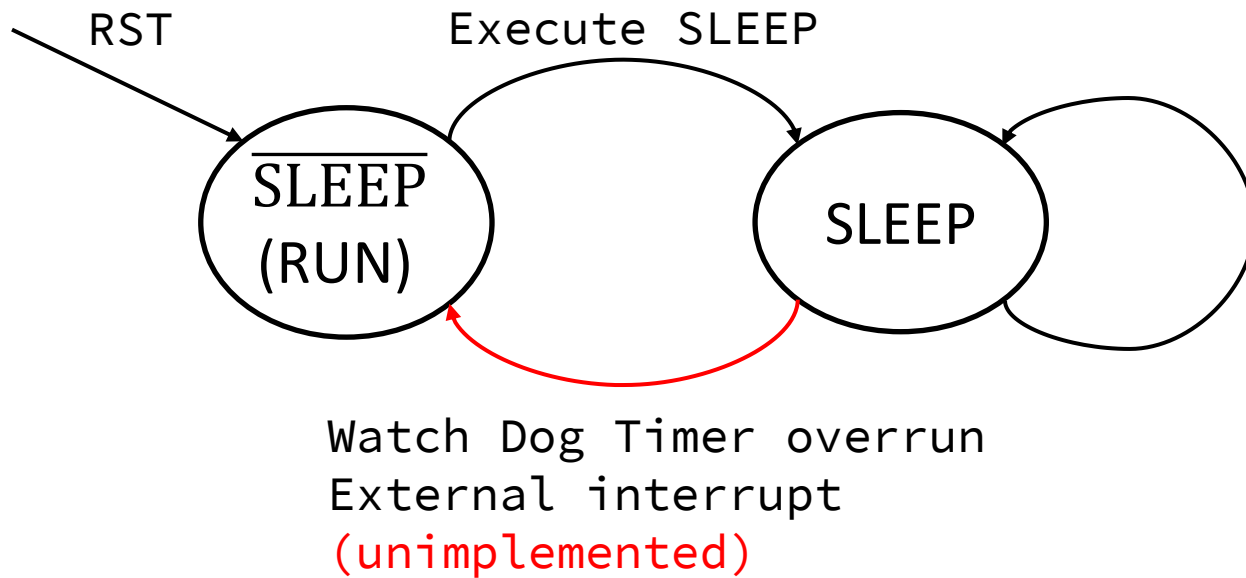
Instantiate for ALU module

```
// Execute  
alu i_alu ( .CLK(CLK), .CB(ALU_CB), .WE(W_W), .B(`IRB),  
            .FI(RDATA),  
            .FO(WDATA),  
            .CI(C), .CO(CO), .DC(DCO), .Z(ZO) );
```



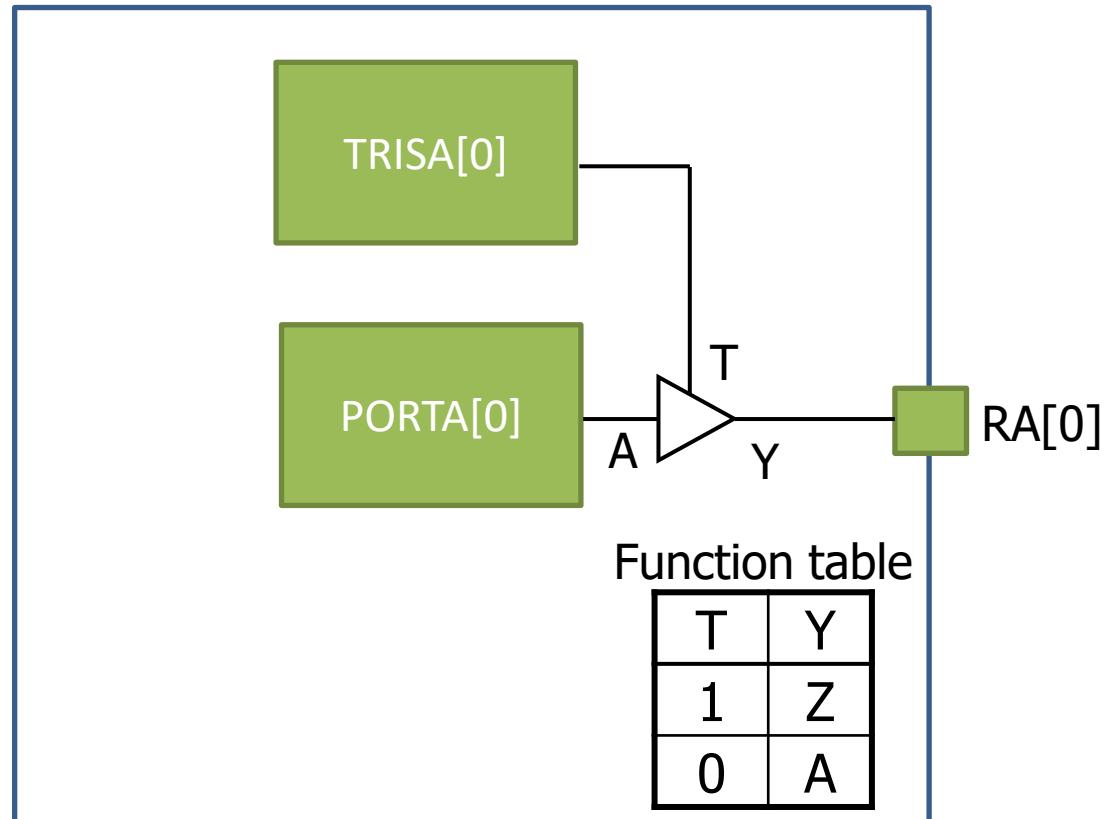
Sleep mode

```
// Sleep mode
always @( posedge CLK or posedge RST )
begin
    if(          ) SLEEP <=    ; else
    if(          ) SLEEP <=    ;
end
```



Tristate Buffer for GPIO

```
//  
// Tristate buffer for GPIO  
//  
assign RA[0] = ( TRISA[0] ) ? 1'bZ : PORTA[0];  
...  
  
assign RB[0] = ( TRISB[0] ) ? 1'bZ : PORTB[0];  
...
```



Top module: pic16.v

```
// Micro Controller based on Microchip PIC16
// for Digilent NEXYS4 DDR Board
module pic16 ( CLK, nRST, RA, RB );
    input          CLK, nRST;
    inout [7:0]    RA;  // RA has 8-bit width same as PIC16F648A
    inout [7:0]    RB;
```

```
parameter PROG = "program.mem";
```

```
// Reset
```

```
wire          RST, nPIC_RST, PIC_RST;
assign RST = ~nRST;
```

DCM module is automatically generated
by Clock module generator on Vivado.

```
// Clock Module for PIC clock
wire          PICCLK, CLK100;
pic_dcm
// #( .CLKOUT1_DIVIDE(10), .CLKFBOUT_MULT_F(8.000) ) // 100*8.00/10=80MHz(12.50ns) 7z020:NG
// #( .CLKOUT1_DIVIDE( 8), .CLKFBOUT_MULT_F(6.000) ) // 100*6.00/ 8=75MHz(13.33ns) 7z020:NG
// #( .CLKOUT1_DIVIDE(10), .CLKFBOUT_MULT_F(7.000) ) // 100*7.00/10=70MHz(14.29ns) 7z020:OK
// #( .CLKOUT1_DIVIDE( 9), .CLKFBOUT_MULT_F(6.000) ) // 100*6.00/ 9=66MHz(15.00ns) 7z020:OK
// #( .CLKOUT1_DIVIDE(15), .CLKFBOUT_MULT_F(9.750) ) // 100*9.75/15=65MHz(15.38ns) 7z020:OK
// #( .CLKOUT1_DIVIDE(10), .CLKFBOUT_MULT_F(6.000) ) // 100*6.00/10=60MHz(16.67ns) 7z020:OK
dcm0 ( .CLK_IN1(CLK), .CLK_OUT1(CLK100), .CLK_OUT2(PICCLK), .RESET(RST), .LOCKED(nPIC_RST) );
```

```
assign PIC_RST = ~nPIC_RST;
```

```
pic16core #( .PROG(PROG) )
```

```
    i_pic16core ( .CLK(PICCLK), .RST(PIC_RST), .RA(RA), .RB(RB) );
```

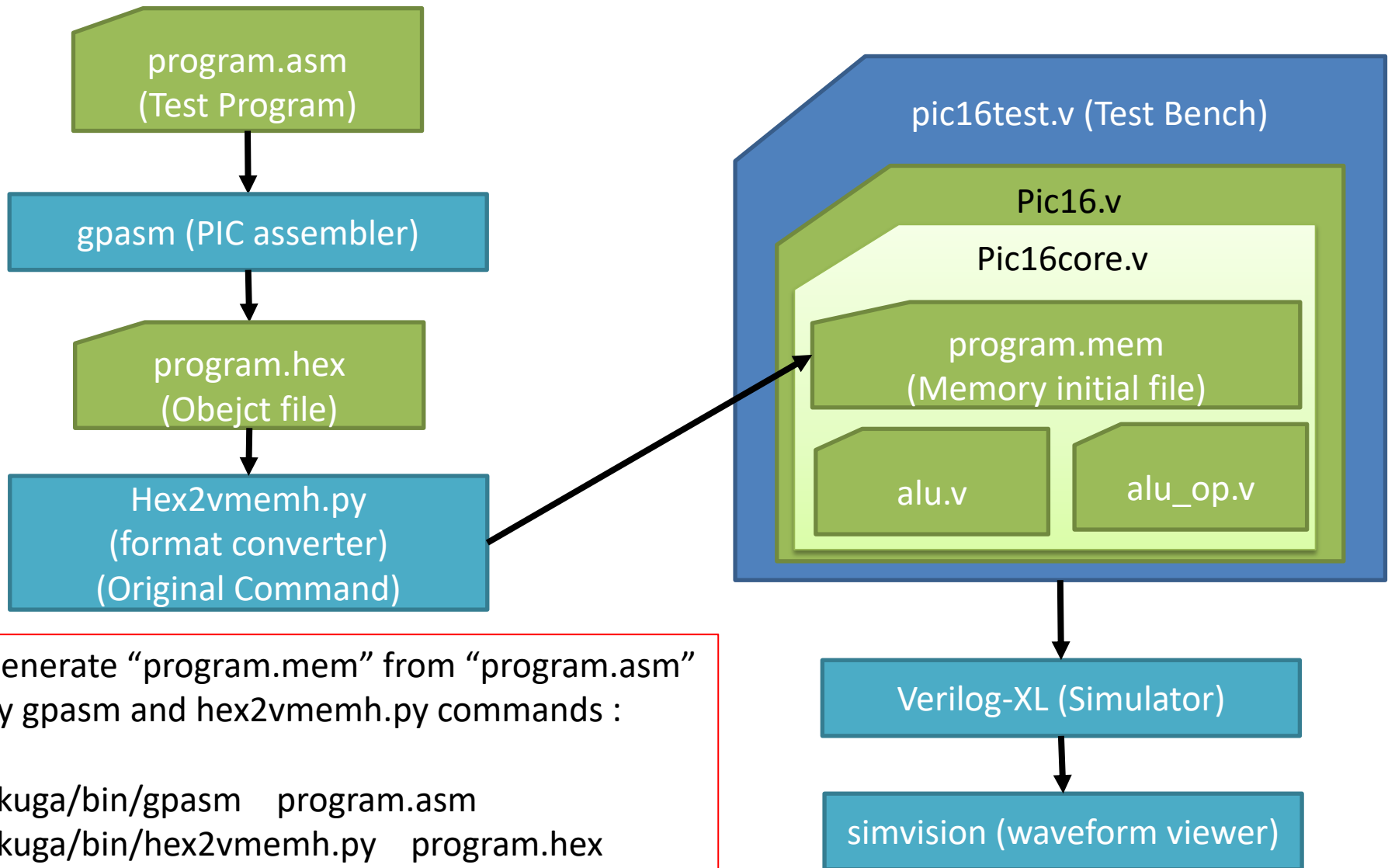
```
endmodule
```

} Instantiation

Processor Simulation

DESIGN EXAMPLE

Processor Simulation



Test Bench: pic16test.v

```
`timescale 1ns/1ns
module pic16test;
    reg          CLK, nRST;
    wire [7:0] RA, RB;

    initial
    begin
        CLK = 0;
        while(1) CLK = #5 ~CLK;
    end

    initial
    begin
        nRST = 0;
        nRST = #100 1;
        #200000;
        $finish;
    end

    data_alias da ( RA, 8'ha5 );

    initial
    begin
        $shm_open("waves.shm");
        $shm_probe("as");
    end

    pic16 #( .PROG("program.mem") )
        i_pic16 ( .CLK(CLK), .nRST(nRST), .RA(RA), .RB(RB) );
endmodule
```

module data_alias(alias_sig_o, alias_sig_i);
 input [7:0] alias_sig_i;
 output [7:0] alias_sig_o;

 assign alias_sig_o = alias_sig_i;
endmodule

**Clock
100MHz**

**reset
and
simulation
time**

Alias assignment for inout port

**Directives for signal monitoring
by verilog-XL and simvision, Cadence EDA tools**

Instantiation

Test program: program.asm

```
LIST      P=PIC16F84A
INCLUDE "p16f84a.inc"

        bsf      STATUS, RP0      ; Set RP0 (Select Bank 1)
        clrw                      ; clear W
        movwf    TRISB            ; TRISB = 00h
        bcf      STATUS, RP0      ; Clear RP0 (Select Bank 0)

        clrf     H'20'            ; s←0
        movlw    D'10'            ; W←10
        movwf    H'21'            ; i←10
LOOP     movf     H'21',w          ; W←i
        addwf    H'20',f          ; s←i+s
        decfsz   H'21',f          ; i←i-1
        goto     LOOP            ; if NZ loop
        movf     H'20',w          ; W←s
        movwf    PORTB           ; PORTB = W
        sleep
        end
```

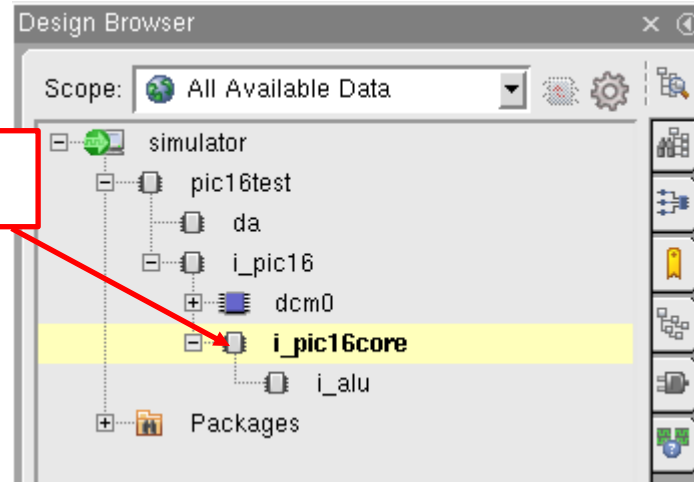
Verilog simulation

- Simulator and waveform viewer are executed on “calc1.st.cs.kumamoto-u.ac.jp”
- For set up the environment, type follows;
ssh -X calc1.st.cs.kumamoto-u.ac.jp
source ~kuga/setup/creative2016
- Create “cds.lib” including next line on working directory;
INCLUDE /opt/XILINX/14.7/ncsim/cds.lib
- For the simulation, type follows;
ncverilog pic16test.v pic16.v pic16core.v ¥¥
alu.v pic_dcm.vhd -V93 +nc64bit -gui +access+r

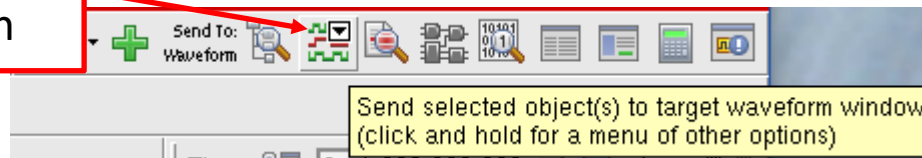
Ncverilog is for the cycle based simulator with mixed languages.
Simvision waveform viewer is automatically invoked by -gui option.

Waveform viewer: simvision

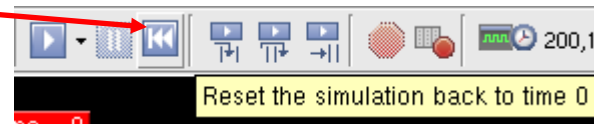
① select "i_pic16core"



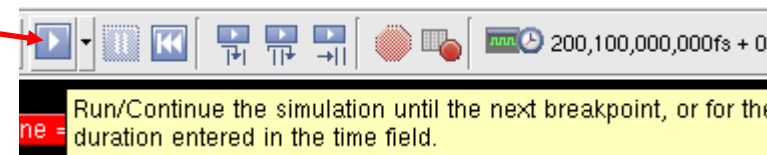
② push this icon



③ Reset simulation



④ Run simulation



Waveform viewer: simvision

Activate RB out

Release RESET

SLEEP mode

Result out

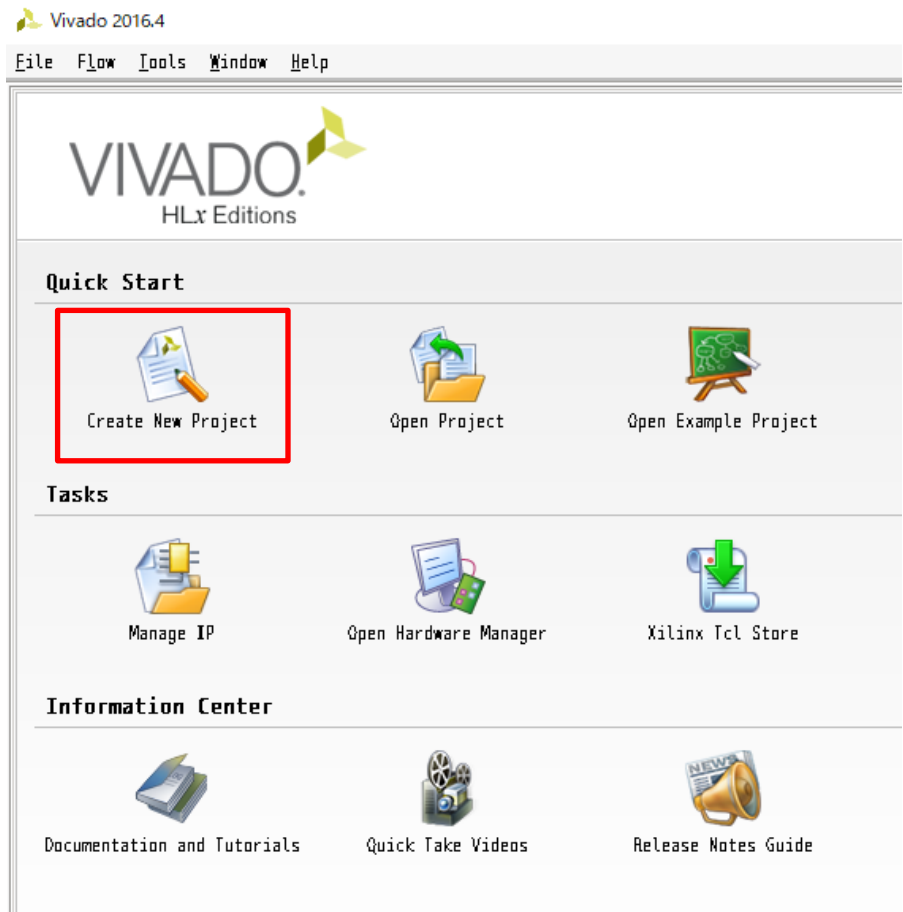


Implementation

DESIGN EXAMPLE

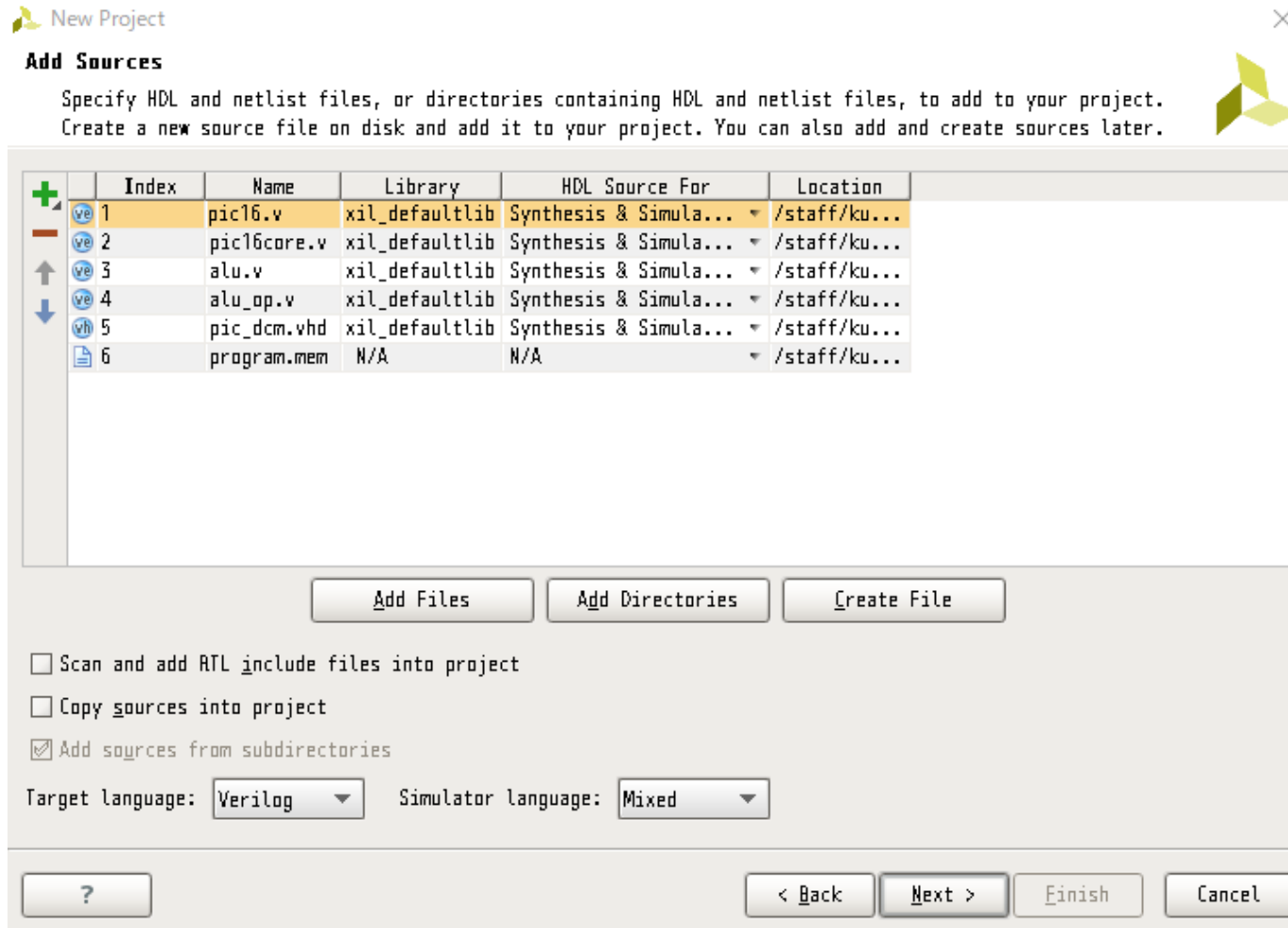
Start the Vivado and Create Project

- To start the implementation tool, type follows;
vivado &
- Click “Create New Project”
 - Click “Next>”
 - Click “Next>” after setting “Project name” and “Project location”
 - Click “Next>” after checking “RTL Project”




Start the Vivado and Create Project

- Continue “Create New Project”
 - Click “Next>” after adding your designed files





Start the Vivado and Create Project



- Continue “Create New Project”
 - Click “Next>” in “Add Existing IP (optional)” without changing
 - Click “Next>” after adding “pic16.xdc” for setting constraints


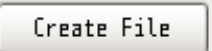
 New Project

Add Constraints (optional)

Specify or create constraint files for physical and timing constraints.

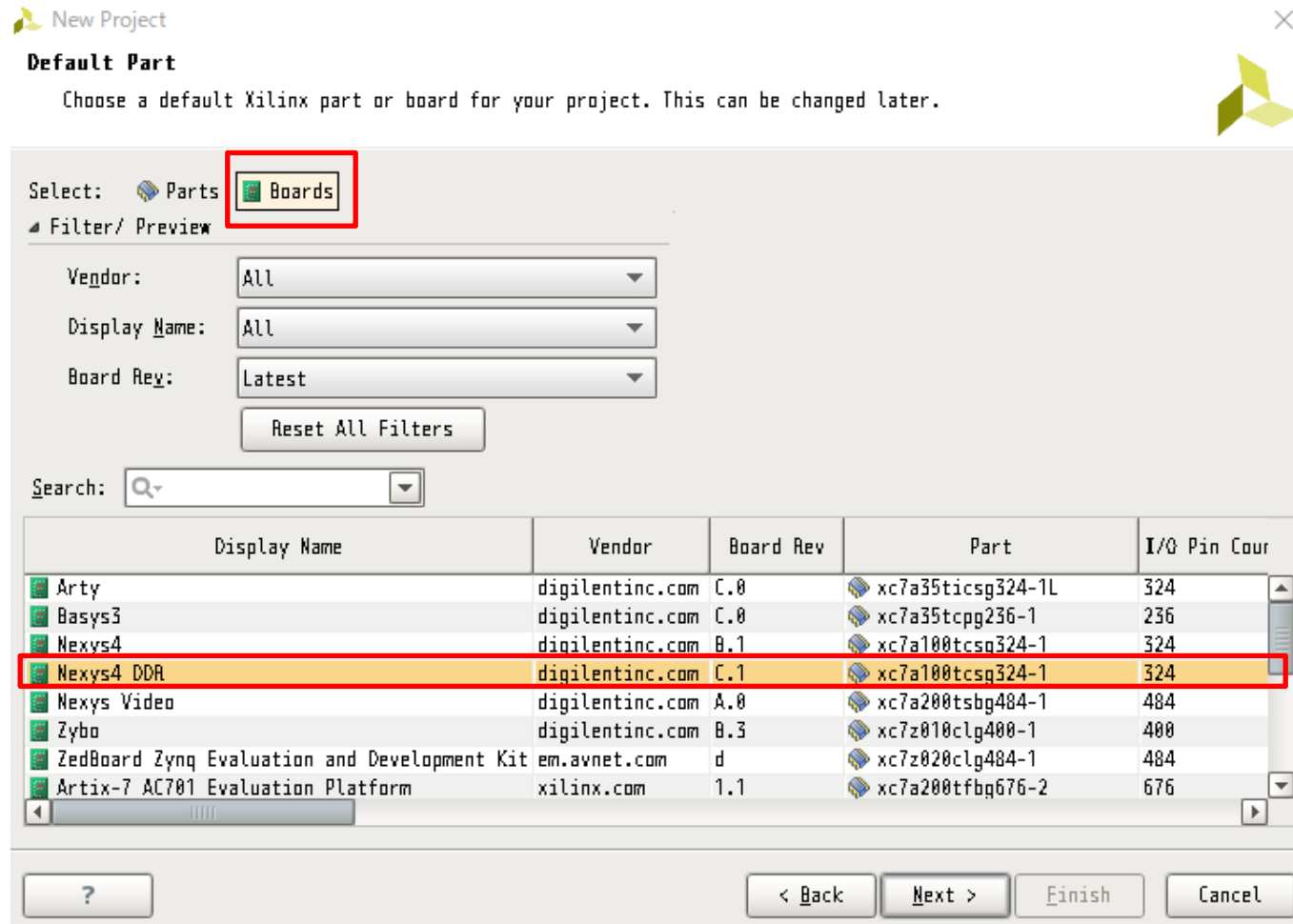
	Constraint File	Location
	 pic16.xdc	/staff/kuga/pic/pic

Start the Vivado and Create Project

- Continue “Create New Project”
 - Click “Next>” after selecting “Nexys4 DDR” board
 - Click “Finish”

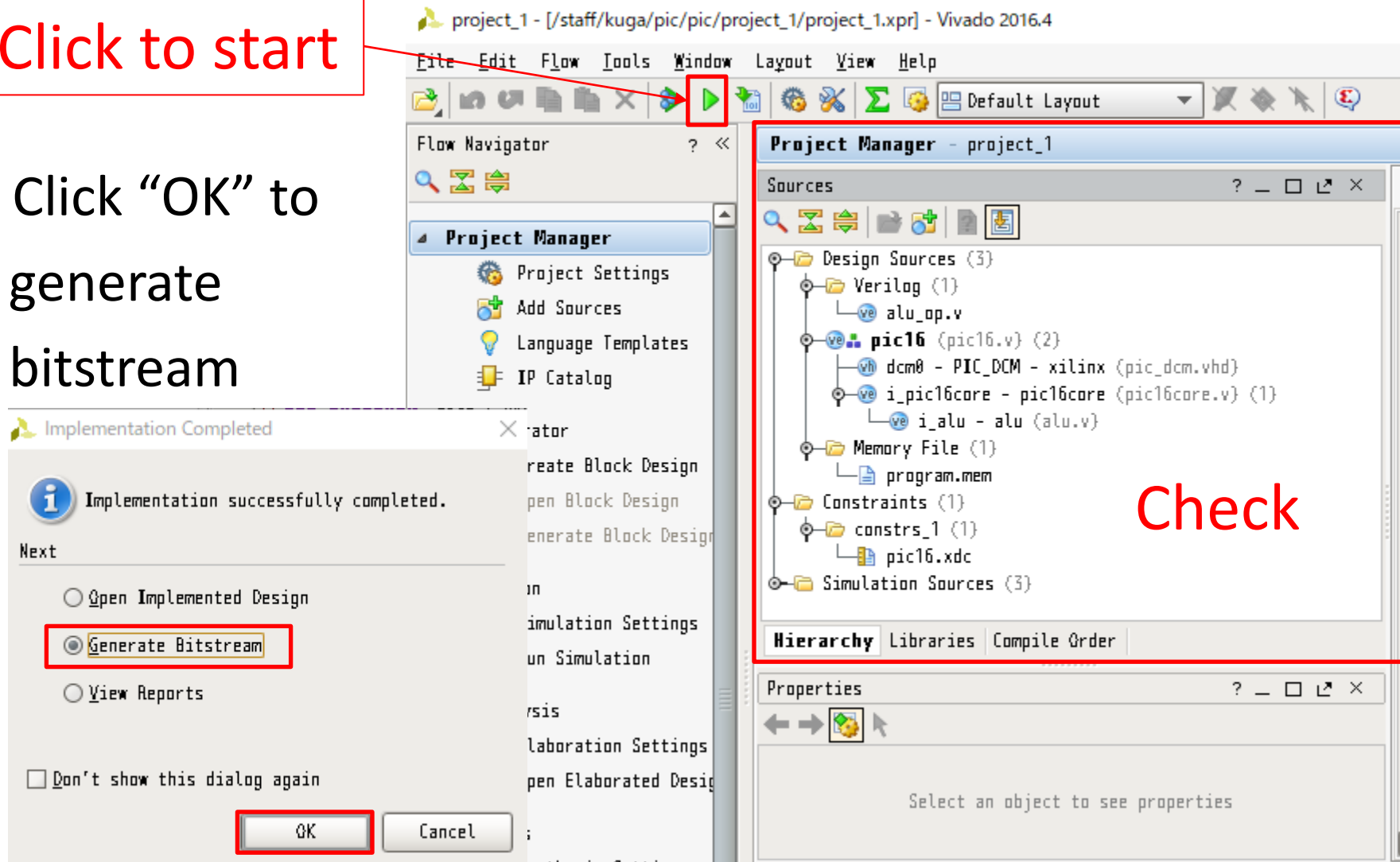


Start Implementation

- Check registered files and start implementation

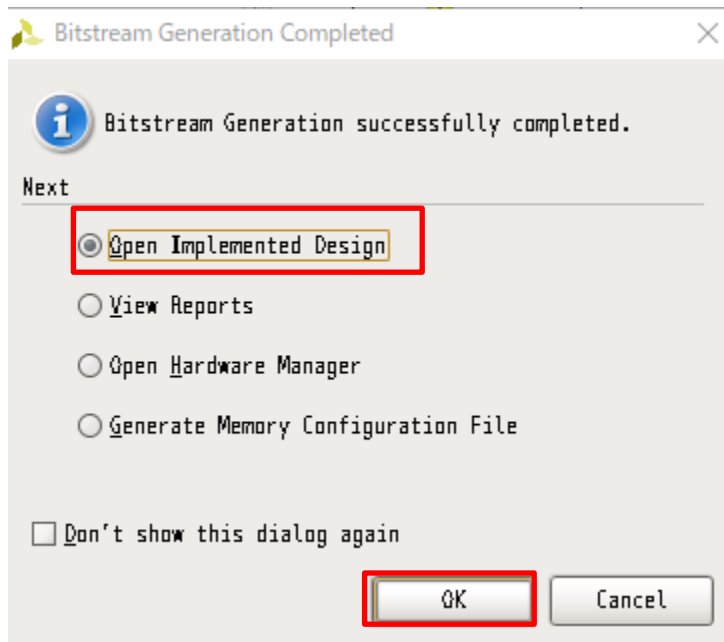
Click to start

- Click “OK” to generate bitstream



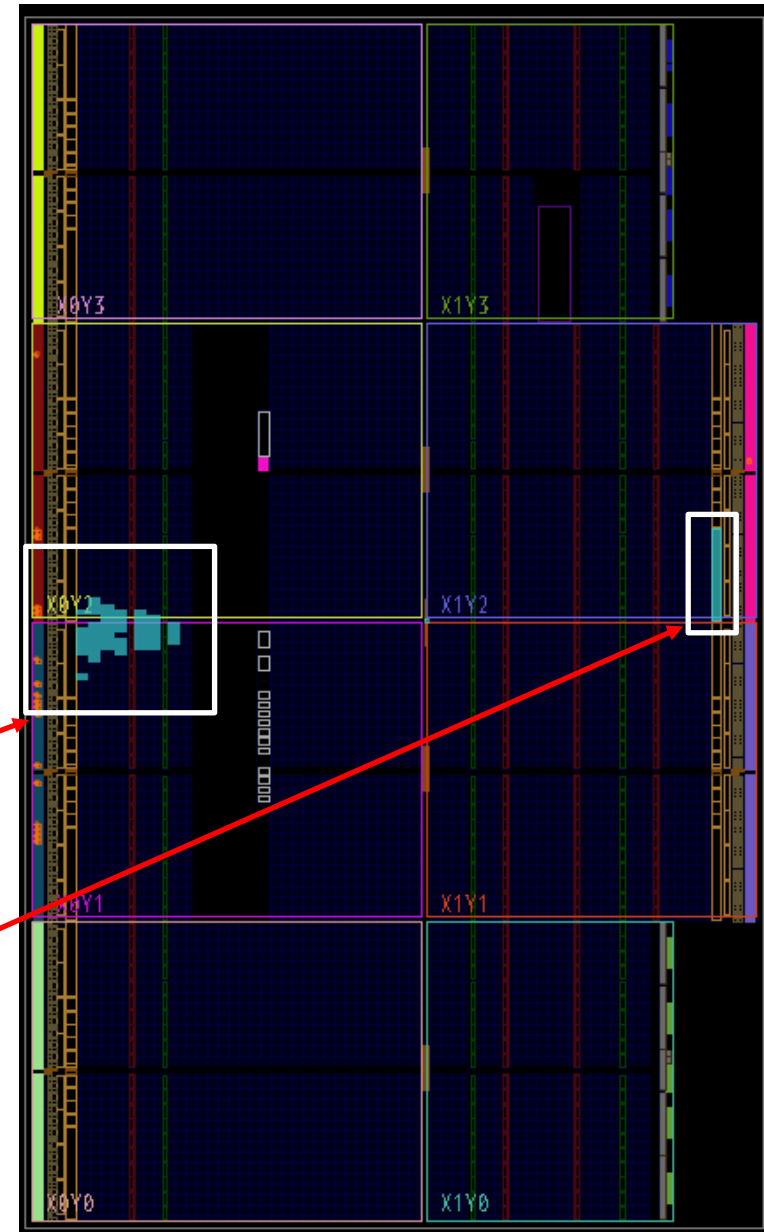
View the Implemented Design

- Click “OK” to open Impl. Design



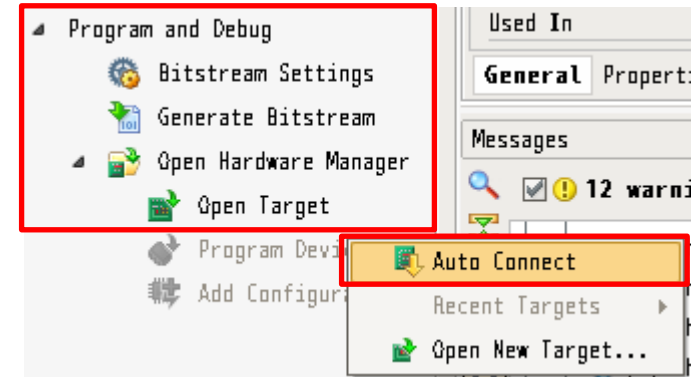
PIC16 microcontroller

Clock module



Check the behavior of your microcontroller

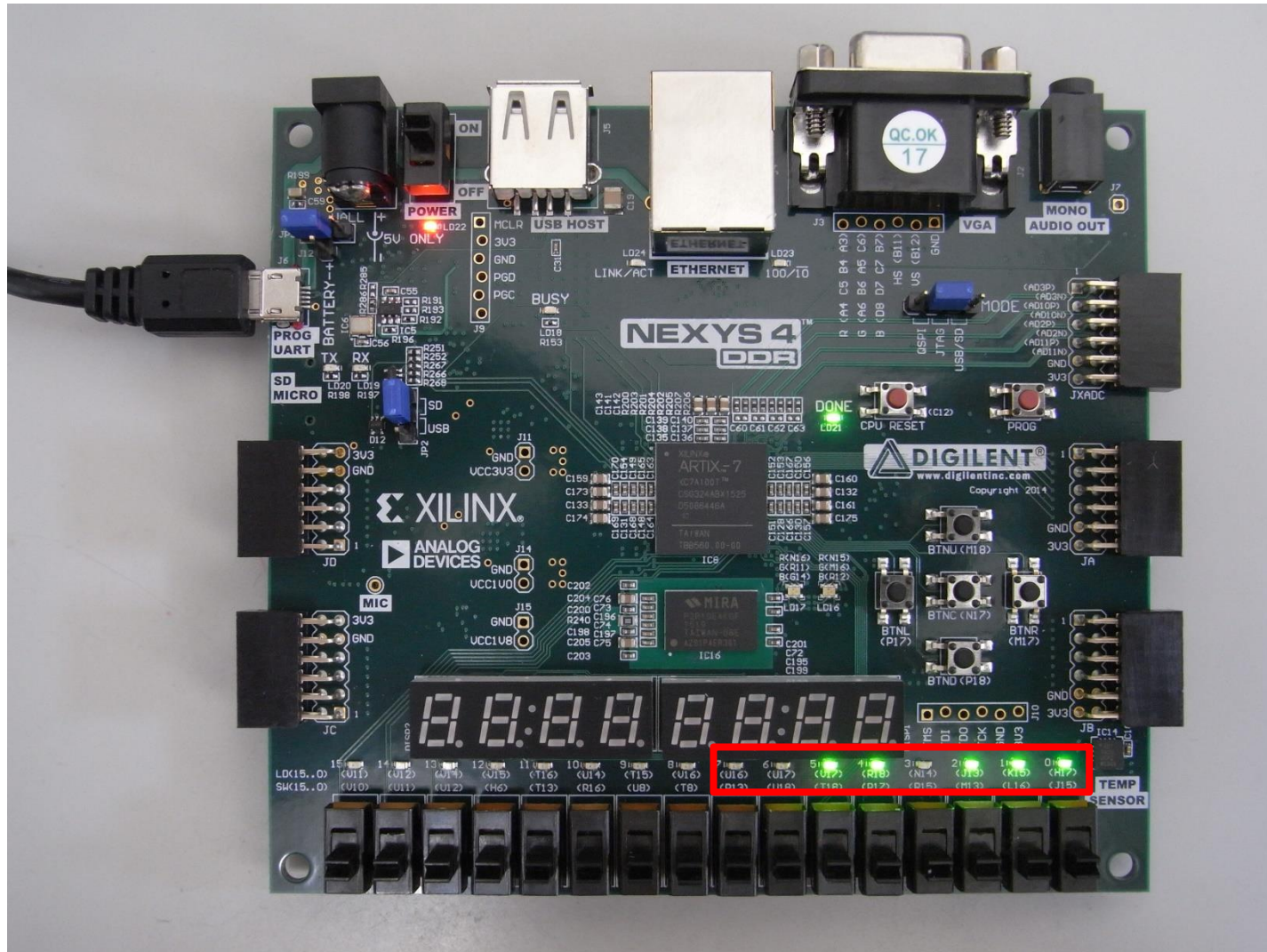
- For download your design, start “Hardware Manager”



- In this class, we don't use “Hardware Manager” because FPGA board does not connect to the host computer.
- To verify your processor behavior, your bitstream file is transfer to local PC with FPGA board.
- Please ask these steps to instructor.
- Your bitstream file is located on your working dir., e.g. “.../project_1/project_1.runs/impl_1/pic16.bit”

Result of Behavior

- 16-LED indicates $(37)_{16}$ for summation result



Try to Design and Implementation
of a PIC16 compatible microcontroller

Let's enjoy design and implementation
for your microcontroller!