EASTERN MEDITERRANEAN UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

1986

CMPE 325 - Computer Architecture II

EXPERIMENT 5

Single Clock DataPath for 16-bit R-type Instructions in ALTERA MAX-PLUS-II VHDL Environment.

Objective: To familiarize with the Single-Clock VHDL implementation for a set of 16-bit R-type instructions, and to measure the response time of several building components in a RISC datapath.

1. Introduction

You have already worked to construct a simple 16-bit instruction set for your project/homework. A typical simple 16-bit instruction set and a corresponding R-type datapath is provided in this experiment. You will be asked to take several measurements that can lead you to a conclusion about the typical speed constraints, and possible improvements of the similar circuits.

In this experiment, we will focus only on the R-type instructions, so that we can observe typical properties of some of the basic building blocks such as the Program-Counter, the Instruction-Memory, the Register-File, and the Adder for updating the Program-Counter.

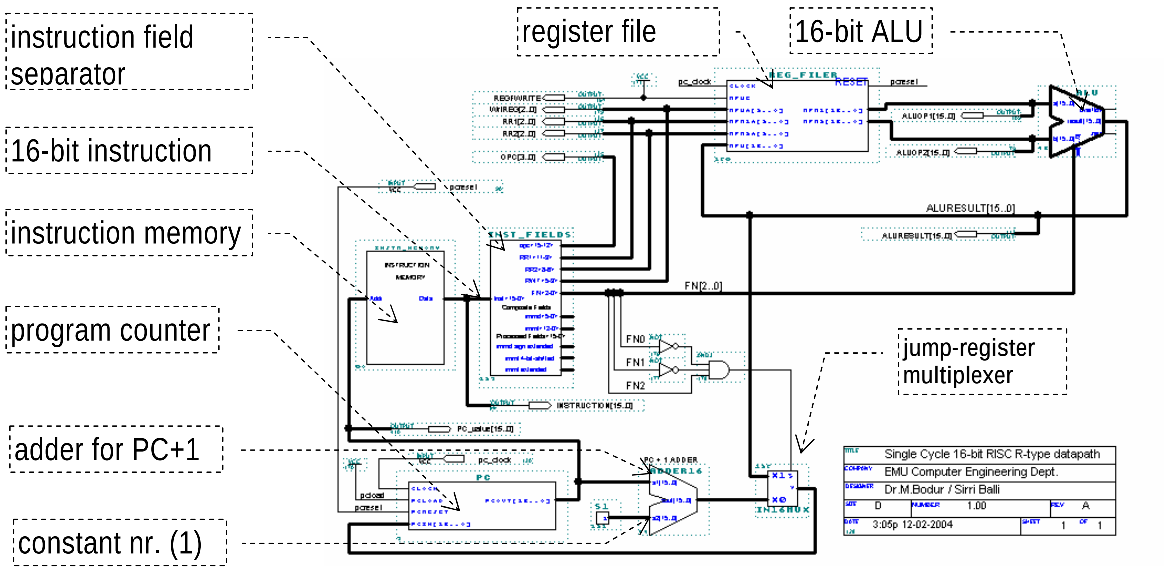


Figure 1-1 Single Cycle DataPath for R-type Instructions.

The complete set of the 16-bit instructions are summarized in the following tables.

Table 1 General representation of R-type Instructions

Opcode	Read1 reg	Read2 reg	Write reg	Function code
4-bits	3-bits	3-bits	3-bits	3-bits

Table 1-1 Karnough Map Representation of the Instruction OpCodes

OpCode	.. 00	.. 01	.. 11	.. 10
00..	R-type	Halt	Goto	Lw
01..	Call (or Jal)	Lui (or LuL)	X	Beq
11..	Addi	X	X	Bneq
10..	Andi	Ori	Xori	Sw

Table 1-2 Function Codes for the R-type Instructions

FNCODE	.00	.01	.11	.10
0..	And	Or	Xor	Add
1..	Jr	ShrA	Slt	Sub

And, our R-type instructions are further listed in the following table.

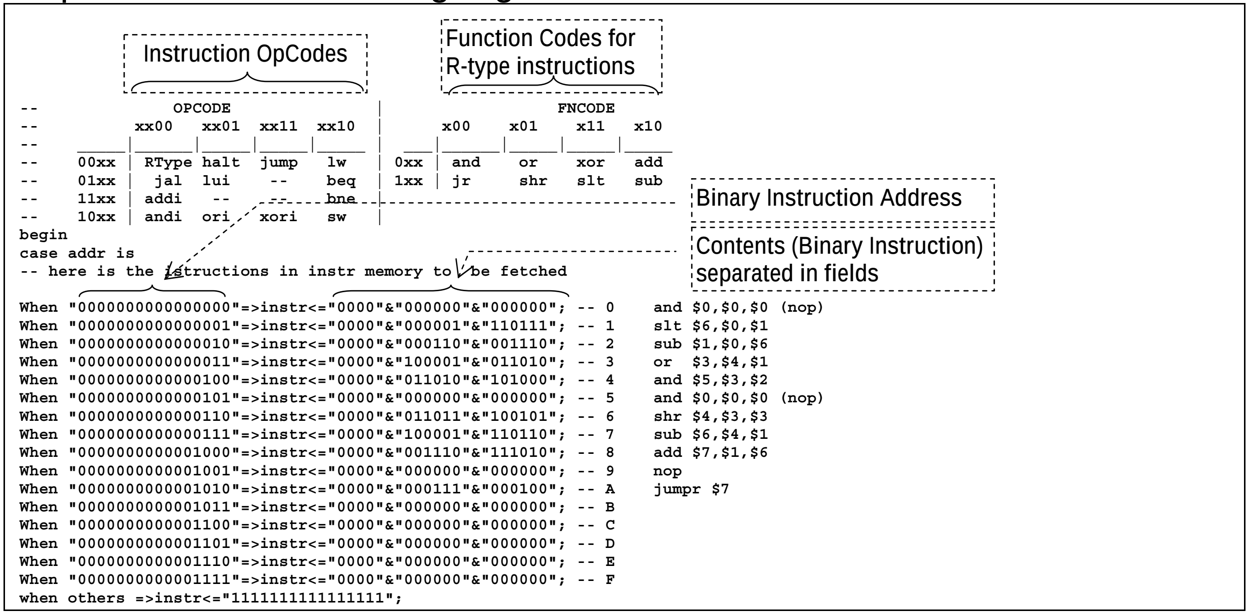
Table 1-3 Representation of R-type Instructions with opcodes

instruction	Opcode	Read-reg1	Read-reg2	Write-reg	Fn code
And	0000	XXX	XXX	XXX	000
Or	0000	XXX	XXX	XXX	001
Add	0000	XXX	XXX	XXX	010
Xor	0000	XXX	XXX	XXX	011
Jump-Register	0000	XXX	000	XXX	100
Shift-right-Arith	0000	XXX	XXX	XXX	101
Subtraction	0000	XXX	XXX	XXX	110
Set less than	0000	XXX	XXX	XXX	111

Note that our instructions have no mnemonics at this stage, because an assembler has not been written yet for this brand new processor. We use the binary codes of the instructions to write them into the instruction memory.

Instruction Memory

You can click-on the instruction memory block to access the contents of the instruction memory. The vhdl file contains sufficient information on how to change the contents of the instruction memory. The instruction memory consists of only 16 instruction words. In the VHDL file, the addresses and corresponding memory contents are written in binary strings. The instruction is written in 4+6+6 character groups for the compatibility with the instruction fields. The four bits of the opcode of all R-type instructions are zero. It is possible to modify the contents of the location by modifying the bit pattern. The contents of the file is explained in the following Figure.


Instruction OpCodes
Function Codes for R-type instructions
Binary Instruction Address
Contents (Binary Instruction) separated in fields

```
--             OPCODE                  |             FNCODE
--          xx00  xx01  xx11  xx10     |      x00    x01    x11   x10
--     _____|_____|_____|_____|_____   |   ___|______|_____|_____|_____
--     00xx | RType halt  jump   lw    |  0xx | and    or    xor   add
--     01xx |  jal  lui    --    beq   |  1xx | jr     shr   slt   sub
--     11xx | addi   --    --    bne   |
--     10xx | andi  ori   xori   sw    |
begin
case addr is
-- here is the istructions in instr memory to be fetched
When "0000000000000000"=>instr<="0000"&"000000"&"000000"; -- 0    and $0,$0,$0 (nop)
When "0000000000000001"=>instr<="0000"&"000001"&"110111"; -- 1    slt $6,$0,$1
When "0000000000000010"=>instr<="0000"&"000110"&"001110"; -- 2    sub $1,$0,$6
When "0000000000000011"=>instr<="0000"&"100001"&"011010"; -- 3    or  $3,$4,$1
When "0000000000000100"=>instr<="0000"&"011010"&"101000"; -- 4    and $5,$3,$2
When "0000000000000101"=>instr<="0000"&"000000"&"000000"; -- 5    and $0,$0,$0 (nop)
When "0000000000000110"=>instr<="0000"&"011011"&"100101"; -- 6    shr $4,$3,$3
When "0000000000000111"=>instr<="0000"&"100001"&"110110"; -- 7    sub $6,$4,$1
When "0000000000001000"=>instr<="0000"&"001110"&"111010"; -- 8    add $7,$1,$6
When "0000000000001001"=>instr<="0000"&"000000"&"000000"; -- 9    nop
When "0000000000001010"=>instr<="0000"&"000111"&"000100"; -- A    jumpr $7
When "0000000000001011"=>instr<="0000"&"000000"&"000000"; -- B
When "0000000000001100"=>instr<="0000"&"000000"&"000000"; -- C
When "0000000000001101"=>instr<="0000"&"000000"&"000000"; -- D
When "0000000000001110"=>instr<="0000"&"000000"&"000000"; -- E
When "0000000000001111"=>instr<="0000"&"000000"&"000000"; -- F
when others =>instr<="1111111111111111";
```

Figure 1-2 The contents of the Instruction Memory.

Instruction field separator

The Instr_Fields block in the lab.gdf Schematic editor is a field-separator block that renames the fields of the instruction (the bit-groups for OPC[3..0], RR1[2..0], RR2[2..0], RWT[2..0], FN[2..0]), and also the immediate and long fields with and without sign extension and 4-bit shift for immediate instruction). The VHDL description for these units are simply a signal connection without any interfacing circuit.

Register File

The register file block reg_filer has been particularly furnished with a synchronous reset input that sets the contents of each register at the positive clock-edge whenever the reset input is high. The setting of i-th register can be modified to any initial value by setting the contents of the `tmp_rf(i)` to a desired value at the then block of the `Reset='1'` condition in VHDL file (you can access it by clicking on the reg_filer block).

```
     if Reset='1' then
                 tmp_rf(1)<= "0000000000010001"; --reg1 = 0x0011
                 tmp_rf(2)<= "0000000000010010"; --reg2 = 0x0012
                 tmp_rf(3)<= "0000000000010011"; --reg3 = 0x0013
                 tmp_rf(4)<= "0000000000010100"; --reg4 = 0x0014
                 tmp_rf(5)<= "0000000000010101"; --reg5 = 0x0015
                 tmp_rf(6)<= "0000000000010110"; --reg6 = 0x0016
                 tmp_rf(7)<= "0000000000010111"; --reg7 = 0x0017
```

Figure 1-3 Initial values of the register_file contents.

ALU

The ALU block in the lab.gdf graphic file is written in VHDL code. The 'case' statement in the VHDL code corresponds to a multiplexer, and the add-sub functions are optimized by the MAXPLUS2 compiler to the most compact form.

```
begin
case sel is
When "000" => temp(15 downto 0)<=a and b; temp(16)<='0';-- and operation
When "001" => temp(15 downto 0)<=a or b;temp(16)<='0';  -- or operation
When "010" => temp<=a+b; -- add operation
When "011" => temp(15 downto 0)<=a xor b;temp(16)<='0'; -- xor operation
When "100" => temp<=a+b; -- add operation
When "101" => temp(14 downto 0)<=a(15 downto 1);temp(15)<=a(15); -- shra
When "110" => temp<=a+not(b)+1; -- subtract operation
When "111" => temp1<=a+not(b)+1;temp <= "0000000000000000" & temp1(15); -- slt
when others => temp<= "XXXXXXXXXXXXXXXX"; -- actually there is no such a case
end case;
```

Figure 1-4 VHDL code of 16-bit ALU

With these ALU operation ("sel") codes we can connect the `FN[2..0]` field of the instruction directly to the `sel[2..0]` inputs of ALU to determine its function. The inputs and output of ALU is furnished with output ports ALUOP1[15..0], ALUOP2[15..0] and ALURESULT[15..0] for monitoring purpose in the Waveform Editor.

(About Project/HW-2: In your second Project Homework-2 you have been asked to design this unit using the graphical schematic capture. You can install your unit into the datapath to test how it works in the datapath).

Jump Register Multiplexer

Finally, a multiplexer provides a datapath from ALURESULT to PC data input PCin[15..0] for the implementation of the jump-to-register instruction. A patch decodes the FN code for JR instruction code, and connects the ALURESULT to PCIN to set the next instruction address from the (ALUop1 + ALUop2).

Dependency analysis

In the R-type datapath the following time instances are observable:

a- Clock-cycle

(T_C: from clock-edge to clock-edge)

b- iPC+1 calculated by adder

(T_{nPC}: from nPC to stabilization of PCin),

c- iPC+1 $\rightarrow$ nPC ,

(T_{PC}: from clock-edge to stabilized nPC),

d- Instruction memory access

(T_{IM}: from stabilized nPC to stabilized instruction)

e- instruction separated to the fields,

(T_{IF}: from stable instruction to stable fields)

f- Reg[rr1] and Reg[rr2] becomes stable,

(T_{RR}: from stable fields to stable register contents) g- ALU produce the result of the operation,

(T$_{ALU}$: from stable ALUinputs to stable ALUresult)

h- ALUresult is written to Reg[rwt],

(not observable, around 2ns)

The following two constraints must be satisfied in all conditions (including the worst conditions) for this datapath to work properly.

1- $\mathbf{T_C > T_{nPC}}$ **;**

2- $\mathbf{T_C} >$ $T_{PC} + T_{IM} + T_{IF} + T_{RR} + T_{ALU}$;

2. Experimental Practice

On this single-cycle-R-type datapath we will observe several properties of a single clock implementation.

1-We expect that ALU give the longest delay when adding FFFF +1, where a carry propagates through the carry. Set the first three instructions in the instruction memory to

```
When "0000000000000000"=>instr<="0000"&"000000"&"000000";-- 0 and $0,$0,$0 (nop)
When "0000000000000001"=>instr<="0000"&"000001"&"110111";-- 1 slt $6,$0,$1
When "0000000000000010"=>instr<="0000"&"000110"&"001110";-- 2 sub $1,$0,$6
When "0000000000000011"=>instr<="0000"&"001110"&"001010";-- 3 add $1,$1,$6
```

Then save the instruction memory VHDL code, and compile the project. Run the simulator (grid size 2ns, pc_clock multiplied with 40) and open the SCF file (step sizeto watch the Waveforms.

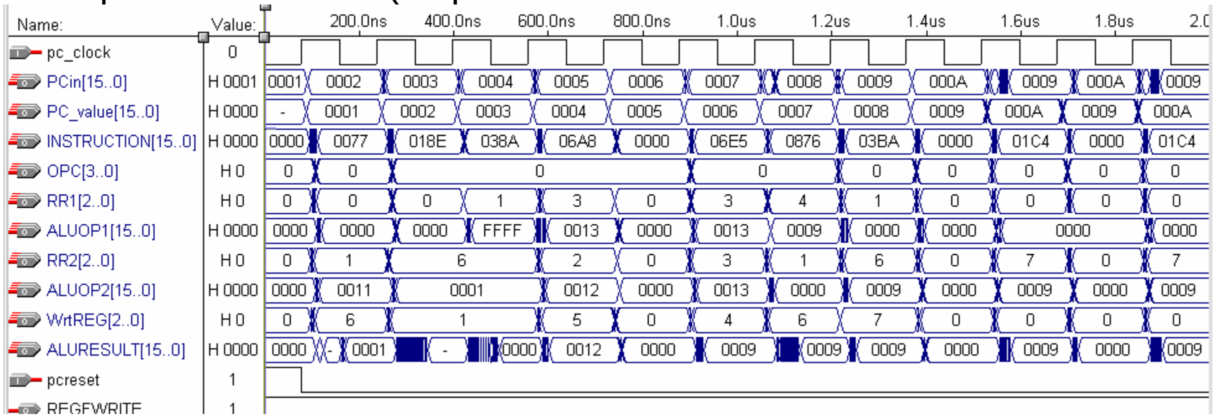


Figure 2-1 Overall view of the execution of R-type instructions in SCF

When you zoom to 240ns you will see the following waveforms

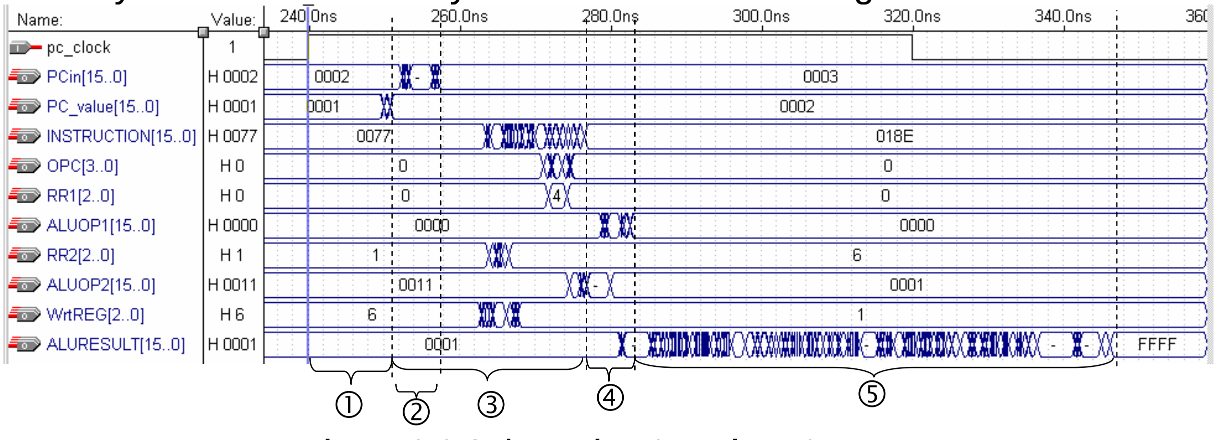


Figure 2-2 Subtraction 0–1 gives 0xFFFF

In this waveform diagram
①=$\mathbf{T_{PC}}$; ②=$\mathbf{T_{nPC}}$; ③=$\mathbf{T_{IM}}$; ④= $\mathbf{T_{IF}}$; ④= $\mathbf{T_{RR}}$; ⑤= $\mathbf{T_{ALU}}$
Similarly, you will observe on the same waveform at time=400 addition 0xFFFF+1=0x0000, which also requires 32 carry propagation.

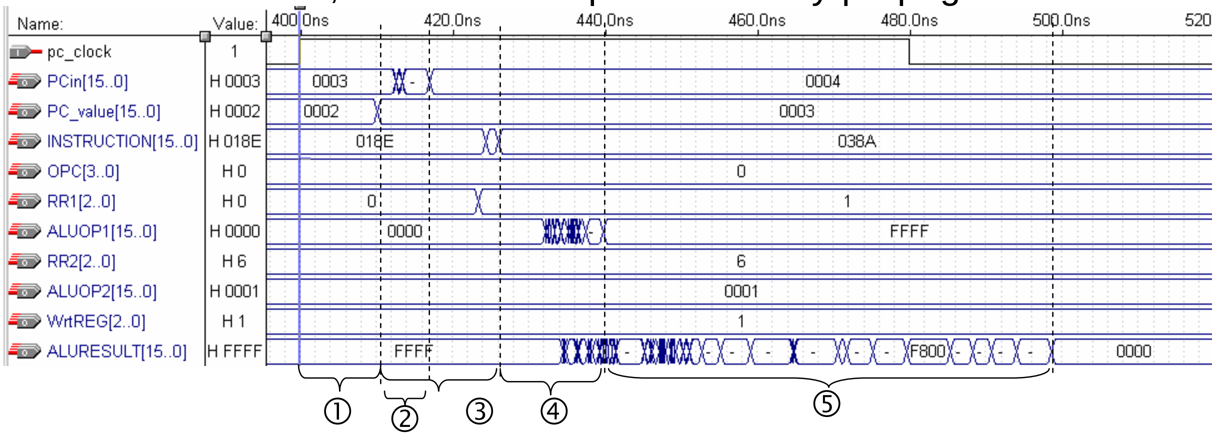


Figure 2-3 Addition 0xFFFF+1 gives 0.

2. On these two waveform charts read the delays of the following operations into the report-sheet.
①=T_{PC} ; ②=T_{nPC} ; ③=T_{IM} ; ④= T_{IF} ; ④= T_{RR} ; ⑤= T_{ALU}

3. Zoom in to the range 1.20 μs to 1.28 μs where a jump-register instruction is in execution. Explain each step of the execution into the report sheet
(i.e.,
at 1211ns PC is stabilized to 0x0008,
at 1222ns nPC=iPC+1 is stabilized to 0x0009,
... etc)

Name: ___________________ Student Number:_____________
Submitted to (Asst.): ___________________ Date:dd/mm/yy ___/___/___

EASTERN MEDITERRANEAN UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

1986

CMPE 325 - Computer Architecture II
EXPERIMENT 5- Reporting Sheet

Section 2.2

	at 240ns	at 400ns	maximum value
T_C:			
T_{nPC}:			
T_{PC}:			
T_{IM}:			
T_{IF}:			
T_{RR}:			
T_{ALU}:			

What is the minimum possible clock period Tcmin? **Tcmin=**......

What is the maximum possible clock rate Fcmax? **Fcmax=**......

Section 2.3

Explain what happens in between the time interval from 1200ns to 1280ns:

Grading: Lab Performance:
Asst. Observations: