CMPE 325 - Computer Architecture II

EXPERIMENT 6

Complete Single Clock DataPath for 16-bit 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

In this experiment you will work on a single clock cycle 16-bit datapath that can perform R-type, I-type, and L-type instructions.

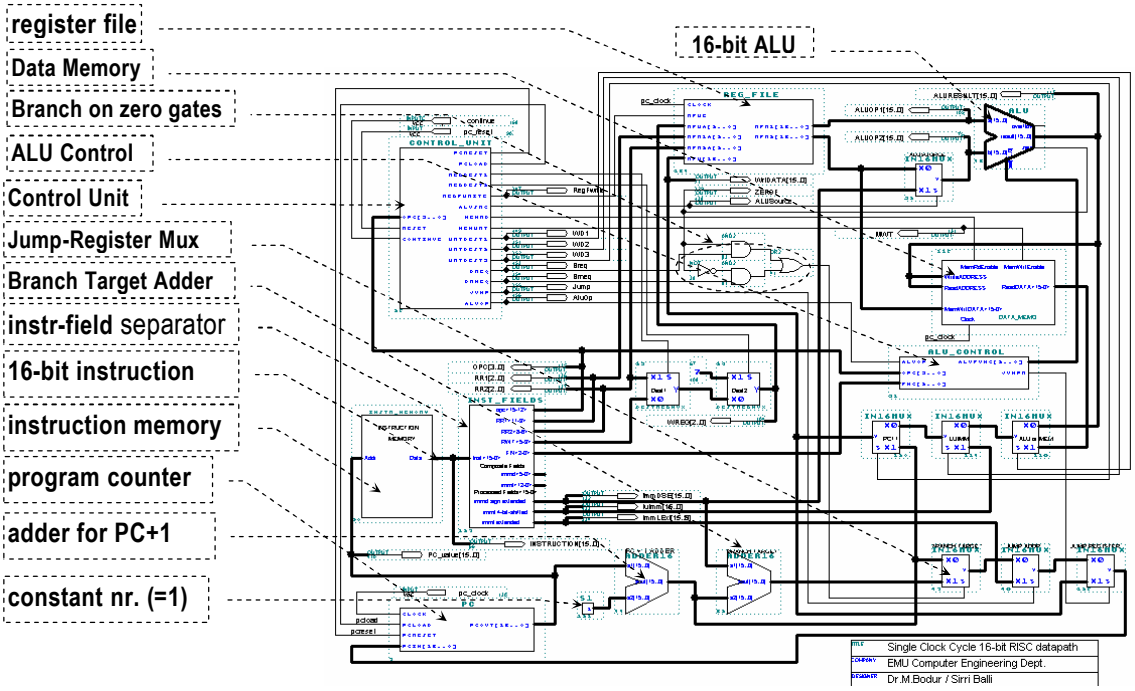

Figure 1 Single Cycle DataPath for all Instructions.

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

Table 1 General representation of R-type Instructions

Opcode	ReadReg1	ReadReg2	WriteReg	FunctionCode
4-bits	3-bits	3-bits	3-bits	3-bits

Table 2 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 3 Function Codes for the R-type Instructions

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

The **R-type** instructions **And, Or**, and **Xor** operates bitwise. **Add** and **Sub** works on signed integers. **ShrA** is one bit shift, and shifts A-input of the ALU one bit right by keeping the sign-bit. **Slt** sets the destination register to one if the first source is less than the second source. **Jr** is the jump return instruction, it is detected by jump-return-gates in the ALU control, and connects the ALU-result to the PC-in, so that with the clock-edge PC-value changes to source1+source2.

The **I-type** instructions **Addi, Andi, Ori, Xori, Lw, Sw, Beq, and Bneq** have a 6-bit immediate operand, which is sign extended in the field-separator block to a 16-bit immediate value.

The **L-type** instruction **Goto** has 12-bit Long-immediate field for the address. **Call** and **Lui** are also L-type, and they use register-7 for destination (implied addressing).

Halt instruction stops the execution of the program until an external signal "continue" goes high.

Instruction Memory

You can click-on the **instruction memory** block to access the contents of the instruction memory. 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 Figure 2.

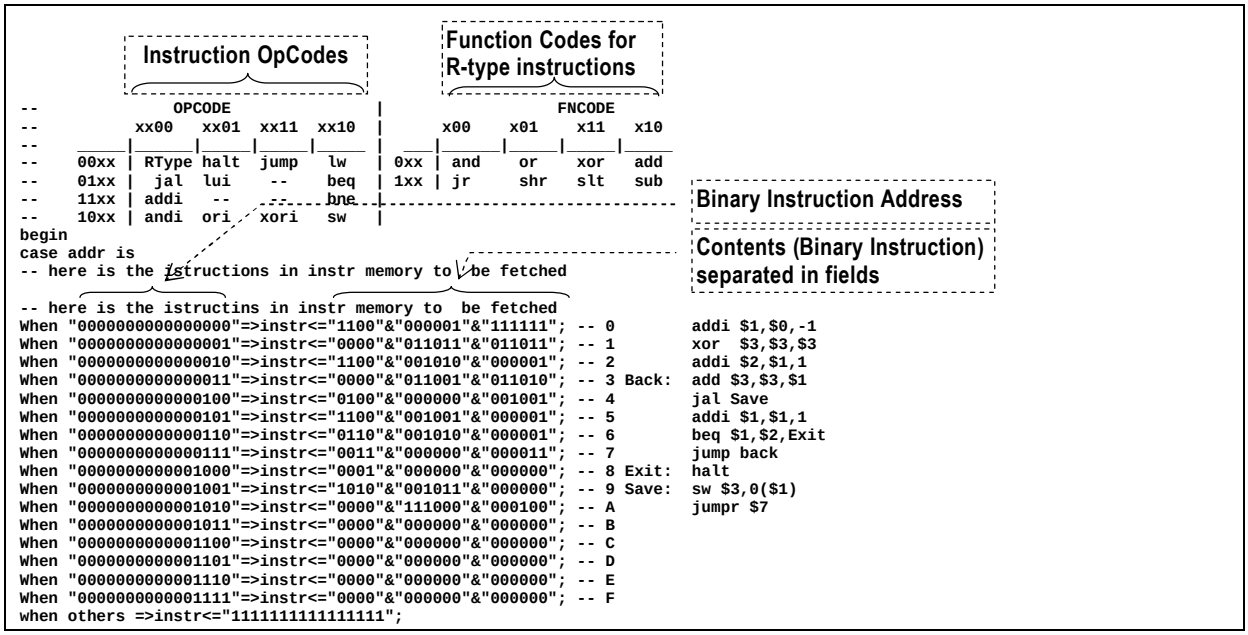

Figure 2 The contents of the Instruction Memory.

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 contains total 7 registers and a constant zero at address 0. It has two read-register-file (RR1, RR2), one write-register-file (WReg) and a 16-bit write-data input. RR1, RR2 and WReg specify the address of the register to access. Writing a data occurs at the positive clock-edge, while reading a register is combinatory (does not require a clock-edge).

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. The ALU-operation select codes "**sel**" are given in

Table 4 Select codes of 16-bit ALU functions

Sel. code	000	001	010	011	100	101	110	111
Function	and	or	add	xor	add	sra	sub	slt

With these ALU function (**"sel"**) codes the ALU-control unit can transfer the **`FN[2..0]`** field of the instruction directly to the **`sel[2..0]`** inputs of **ALU**. 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.

Dependency analysis

In the execution of the load-word instruction (the longest of all) the following time instances are observable:

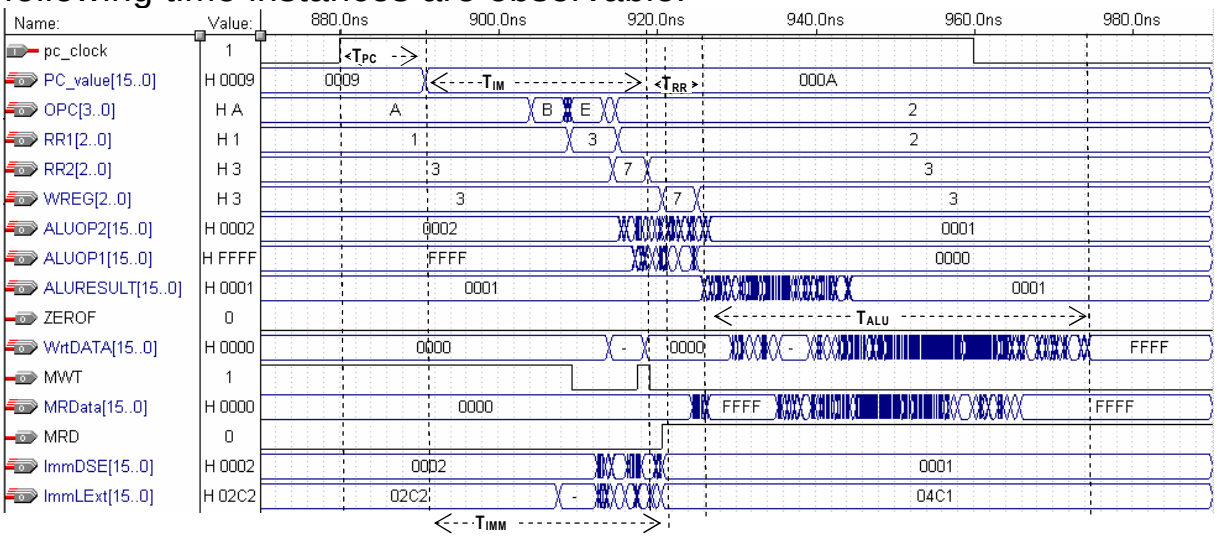

Figure 3 Important Time Intervals for time dependency analysis

a- Clock-cycle
(**T_C**: from clock-edge to clock-edge)

c- iPC+1 → nPC , state change propagation time
(**T_{PC}:** from clock-edge to stabilized nPC),

d- Instruction memory access
(**T_{IM}**: from stabilized nPC to stabilized instruction)

d- Immediate field stabilization interval
(**T_{IMM}**: from stabilized nPC to stabilized immediate field)

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],
(T_{MWT}: not observable, around 2ns)

The following two constraints must be satisfied in the execution of the LW instructions for this datapath to work properly.

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

2- **$T_C > T_{PC} + Max(T_{IMM}, T_{IM}+T_{RR}) + T_{ALU} + T_{MWT}$;**

You can derive the timing constraints of other instructions (**beq, sw, call**, etc.) similarly by analyzing their waveforms.

2. Experimental Practice

Part-1 LW instruction

In the first part, we will observe the time dependency of the load-word instruction on this single-cycle-16-bit-datapath.

1-We will use 2ns grid-step, and ×40 multiplication factor for the pc_clock generation, so that the clock period will be 160ns. The program to be used in this observation will be

```
0       addi $1,$0,-1
1       xor  $3,$3,$3 (Or addi $3,$0,0)
2       addi $2,$1,1
3 Back: add $3,$3,$1
4       jal Save
5       addi $1,$1,1
6       beq $1,$2,Exit
7       jump back
8 Exit: Halt
9 Save: sw $3,2($1)
A       lw $3,1($2)
B       jumpr $7
```

Check the instruction memory VHDL code, correct it if necessary, and compile the project. Run the simulator (grid size 2ns, pc_clock multiplied with 40) and open the SCF file to watch the Waveforms.

2- The opcode of the LW instruction is 2. Find opcode 2 in the overall diagram, and zoom in the execution of the instruction.

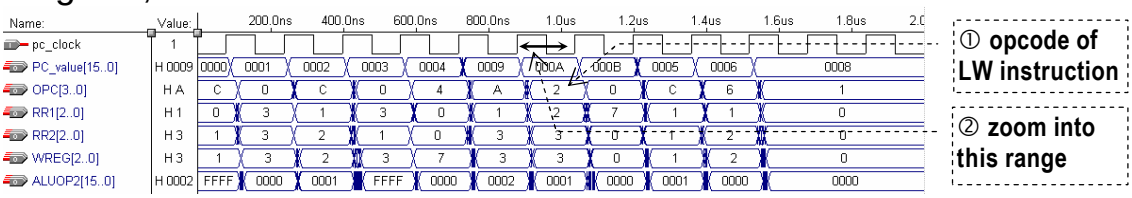

Figure 4 Zoom into execution of the LW instruction

When you zoom to 870-980ns range (key sequence alt-V,T and enter the time values) you will see the waveforms given in Figure 3. Take all necessary measurements to calculate the minimum possible T_C period for this particular case of lw instruction according to the constraints

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

2- $\mathbf{T_C > T_{PC} + Max(T_{IMM}, T_{IM}+T_{RR}) + T_{ALU} + T_{MWT}}$.

Fill in your time interval readings, and calculated minimum TC to the reporting sheet.

Part-2 Branch Instruction

The branch instruction in the program is beq, and has opcode 6. Find the time interval with opcode 6, and zoom into that region (from 1350 to 1500 ns).

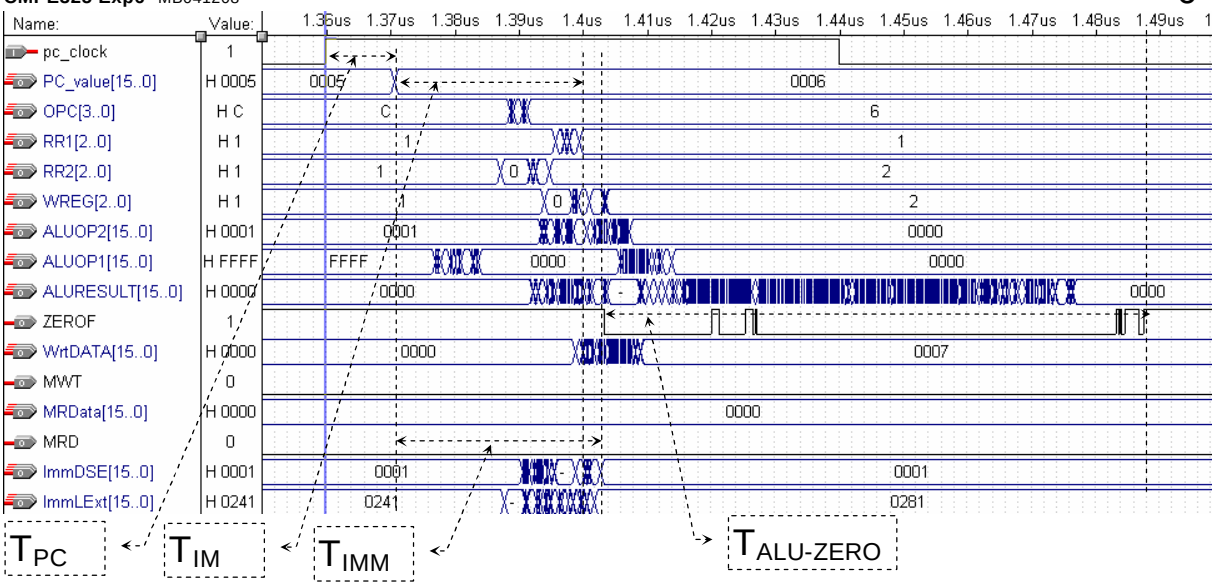

Figure 5 Timing parameters in BEQ instruction.

The time constraint in deciding on the PC-in value is

$$T_{PC} + T_{IM} + T_{RR} + T_{ALU\text{-}ZERO} < T_C$$

A parallel constraint raises up in the target-address calculation path as

$$T_{PC} + T_{IMM} + T_{Branch\text{-}Adder} < T_C .$$

Note that we don't observe the branch target adder output, since we assume that this adder has almost the same delay with the ALU-adder. In your calculation, you can simply assume $T_{Branch\text{-}Adder} = T_{ALU}$.

Read and fill in to the reporting sheet the intervals, and calculate the minimum clock time for this particular branch instruction.

Part-3 Conclusion:

If we complete the dependency analysis for all instructions, we can find a minimum clock time constraint for the worst case of each instruction. Since we don't know which instruction will be executed before the cycle starts, the overall clock time must be long enough for the execution of all instructions. Therefore, we must use the maximum of the minimum-clock-cycles for each instruction. For example, if LW instruction requires minimum 120ns, and BEQ requires minimum 105ns, max(120ns,105ns)= 120 ns will be sufficient both for the execution of the LW and BEQ instruction.

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

EASTERN MEDITERRANEAN UNIVERSITY
COMPUTER ENGINEERING DEPARTMENT

Fall 2ØØ4

CMPE 325 - Computer Architecture II
EXPERIMENT 6- Reporting Sheet

Section 2. Part-1 LW instruction:

	start-time	end-time	interval
T_C			
T_{PC}			
$T_{IM}+T_{RR}$:			
T_{IMM}			
T_{ALU}			

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

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

Section 2. Part-2 Branch Instruction

	start-time	end-time	interval
T_C			
T_{PC}			
$T_{IM}+T_{RR}$:			
T_{IMM}			
$T_{ALU-ZERO}$			
$T_{Branch-Adder}$			

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

Part-3 Conclusion:

If we consider only the LW and BEQ instructions, and assuming that we measured the worst case intervals in part-1 and -2, what is the maximum clock frequency for this datapath.

$F_{C\text{-}Max}$ = ...

Grading: Quiz Performance:
Lab Performance:
Asst. Observations: