EECS150: Spring 2010 Project Checkpoint 1, MIPS150 Processor

UC Berkeley College of Engineering
Department of Electrical Engineering and Computer Science

Revision I

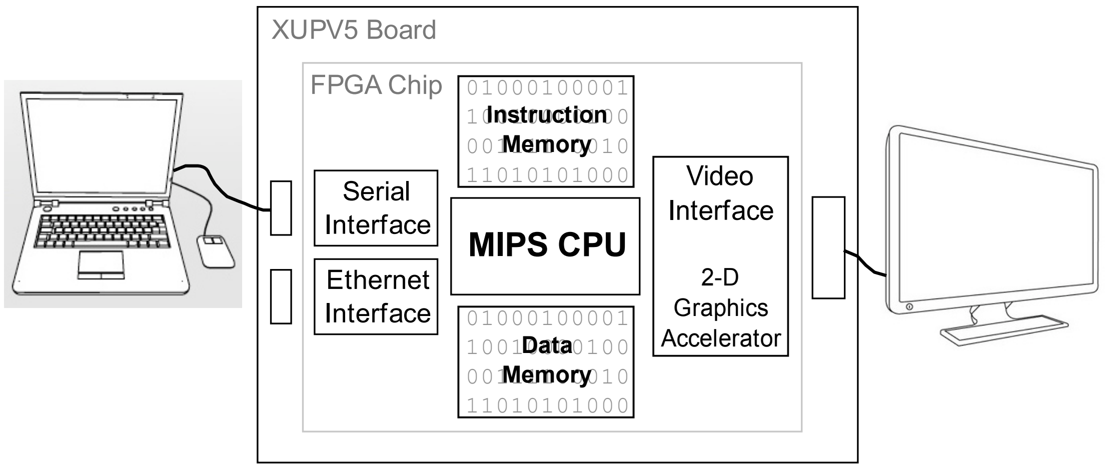
1 Time Table

ASSIGNED	Friday, February 26th
DUE	Week 9: Mar 16th, during your assigned lab section

2 Motivation

You will construct a pipelined (3 stage) implementation of a MIPS reduced instruction set (RISC) architecture. The required instruction set architecture is a strict subset of MIPS (documented in the checkpoint 1 assignment), excluding floating point instructions, traps, misaligned memory accesses, branch and link instructions, and branch likely instructions. Your processor will also exclude coprocessors, as well as a few features normally built into processors to enable operating systems.

Figure 1 EECS150 Spring 2010 Computer System



Assuming your processor is built according to our specifications, you will be able to use a simplified toolflow consisting of a functional simulator, assembler, C compiler, and some library code. The processor will coordinate the function of all other components in your project system. Refer to Figure 1 for a bird's-eye view of the processor's role in this project. Memory mapped I/O (you will implement this later in the semester) will enable the processor to communicate with the other components in your system.

Although the design and implementation of this system are entirely up to you, we impose a small set of requirements in order to better help you succeed in this project. In particular, we enforce a strict requirement of CPI=1 (meaning that no instruction may stall the datapath, and the processor must always perform 1 instruction per cycle). Additionally, you must implement your processor as a 3-stage datapath (detailed in Section 4). The instruction set architecture features a branch/jump delay slot, as well as a delay slot to reduce hardware overhead of guaranteeing one instruction per cycle.

3 Instruction Set

Your processor must implement the full subset of instructions depicted in Table 1. Instructions formats, Opcode's, and Funct fields are from the standard MIPS ISA and have been included for convenience. Note that **we will be using the architected branch, jump, and load delay slots** to reduce hardware overhead.

Table 1 SMIPSv2 Instruction Set

31 26	25 21	20 16	15 11	10 6	5 0	
opcode	rs	rt	rd	shamt	funct	R-type
opcode	rs	rt	immediate			I-type
opcode	target					J-type
		Load and Store Instructions				
100011	base	dest	signed offset			LW rt, offset(rs)
101011	base	dest	signed offset			SW rt, offset(rs)
		I-Type Computational Instructions				
001001	src	dest	signed immediate			ADDIU rt, rs, signed-imm.
001010	src	dest	signed immediate			SLTI rt, rs, signed-imm.
001011	src	dest	signed immediate			SLTIU rt, rs, signed-imm.
001100	src	dest	zero-ext. immediate			ANDI rt, rs, zero-ext-imm.
001101	src	dest	zero-ext. immediate			ORI rt, rs, zero-ext-imm.
001110	src	dest	zero-ext. immediate			XORI rt, rs, zero-ext-imm.
001111	00000	dest	zero-ext. immediate			LUI rt, zero-ext-imm.
		R-Type Computational Instructions				
000000	00000	src	dest	shamt	000000	SLL rd, rt, shamt
000000	00000	src	dest	shamt	000010	SRL rd, rt, shamt
000000	00000	src	dest	shamt	000011	SRA rd, rt, shamt
000000	rshamt	src	dest	00000	000100	SLLV rd, rt, rs
000000	rshamt	src	dest	00000	000110	SRLV rd, rt, rs
000000	rshamt	src	dest	00000	000111	SRAV rd, rt, rs
000000	src1	src2	dest	00000	100001	ADDU rd, rs, rt
000000	src1	src2	dest	00000	100011	SUBU rd, rs, rt
000000	src1	src2	dest	00000	100100	AND rd, rs, rt
000000	src1	src2	dest	00000	100101	OR rd, rs, rt
000000	src1	src2	dest	00000	100110	XOR rd, rs, rt
000000	src1	src2	dest	00000	100111	NOR rd, rs, rt
000000	src1	src2	dest	00000	101010	SLT rd, rs, rt
000000	src1	src2	dest	00000	101011	SLTU rd, rs, rt
		Jump and Branch Instructions				
000010	target					J target
000011	target					JAL target
000000	src	00000	00000	00000	001000	JR rs
000000	src	00000	dest	00000	001001	JALR rd, rs
000100	src1	src2	signed offset			BEQ rs, rt, offset
000101	src1	src2	signed offset			BNE rs, rt, offset
000110	src	00000	signed offset			BLEZ rs, offset
000111	src	00000	signed offset			BGTZ rs, offset
000001	src	00000	signed offset			BLTZ rs, offset
000001	src	00001	signed offset			BGEZ rs, offset

The function of each instruction is shown in Table 2.

Table 2 ISA Functional Specification

Instruction	RTL
LW	$R[\$rt] = M[R[\$rs] + SignExt(imm)](delayed)$
SW	$M[R[\$rs] + SignExt(imm)] = R[\$rt]$
ADDIU	$R[\$rt] = R[\$rs] + SignExt(imm)$
SLTI	$R[\$rt] = (R[\$rs] < SignExt(imm))?32'h1 : 32'h0$
SLTIU	$R[\$rt] = (R[\$rs] < SignExt(imm))?32'h1 : 32'h0(unsigned \quad comparison)$
ANDI	$R[\$rt] = R[\$rs] \wedge ZeroExt(imm)$
ORI	$R[\$rt] = R[\$rs] \vee ZeroExt(imm)$
XORI	$R[\$rt] = R[\$rs] \oplus ZeroExt(imm)$
LUI	$R[\$rt] = \{imm, 16'h0\}$
SLL	$R[\$rd] = R[\$rt] << shamt(logical)$
SRL	$R[\$rd] = R[\$rt] >> shamt(logical)$
SRA	$R[\$rd] = R[\$rt] >>> shamt(arithmetic)$
SLLV	$R[\$rd] = R[\$rt] << R[\$rs](logical)$
SRLV	$R[\$rd] = R[\$rt] >> R[\$rs](logical)$
SRAV	$R[\$rd] = R[\$rt] >>> R[\$rs](arithmetic)$
ADDU	$R[\$rd] = R[\$rs] + R[\$rt]$
SUBU	$R[\$rd] = R[\$rs] - R[\$rt]$
AND	$R[\$rd] = R[\$rs] \wedge R[\$rt]$
OR	$R[\$rd] = R[\$rs] \vee R[\$rt]$
XOR	$R[\$rd] = R[\$rs] \oplus R[\$rt]$
NOR	$R[\$rd] = \overline{R[\$rs]} \wedge \overline{R[\$rt]}$
SLT	$R[\$rd] = (R[\$rs] < R[\$rt])?32'h1 : 32'h0$
SLTU	$R[\$rd] = (R[\$rs] < R[\$rt])?32'h1 : 32'h0(unsigned \quad comparison)$
J	$PC = \{(PC)[31:28], JA, 2'b00\}(delayed)$
JAL	$R[31] = PC + 8; PC = \{(PC)[31:28], JA, 2'b00\}(delayed)$
JR	$PC = R[\$rs](delayed)$
JALR	$R[\$rd] = PC + 8; PC = R[\$rs](delayed)$
BEQ	$PC = PC + 4 + (R[\$rs] == R[\$rt])?(SignExt(imm) << 2) : 0(delayed)$
BNE	$PC = PC + 4 + (R[\$rs]! = R[\$rt])?(SignExt(imm) << 2) : 0(delayed)$
BLEZ	$PC = PC + 4 + (R[\$rs] <= 0)?(SignExt(imm) << 2) : 0(delayed)$
BGTZ	$PC = PC + 4 + (R[\$rs] > 0)?(SignExt(imm) << 2) : 0(delayed)$
BLTZ	$PC = PC + 4 + (R[\$rs] < 0)?(SignExt(imm) << 2) : 0(delayed)$
BGEZ	$PC = PC + 4 + (R[\$rs] >= 0)?(SignExt(imm) << 2) : 0(delayed)$

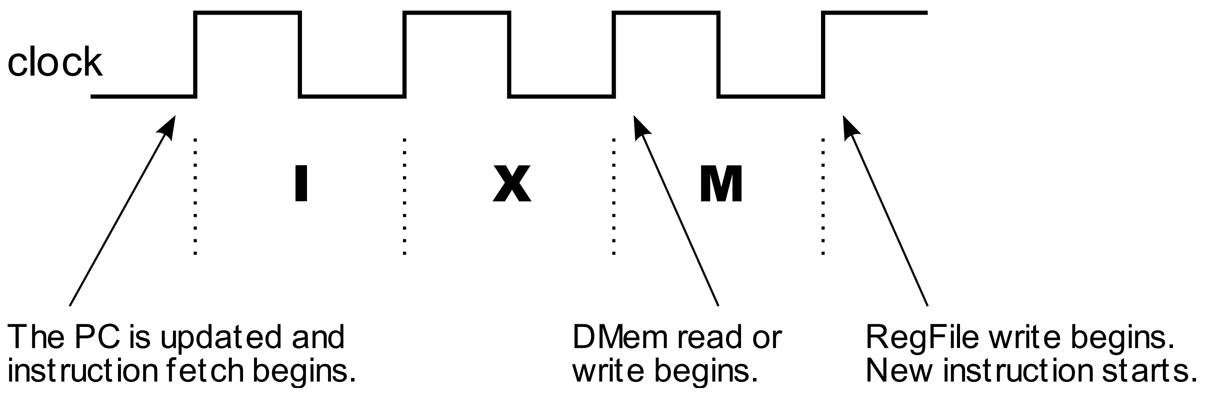
4 Pipeline Stages

As mentioned before, the processor will be required to have 3 pipeline stages. Although we have arbitrarily labeled the stages in lecture as **I**, **X**, and **M** (which stand for Instruction, Execute, and Memory, respectively), it is really up to you what you will be doing in each stage. A simple diagram of what we mean by 3 pipeline stages is shown in Figure 2. **Note that a 3 stage pipeline is defined by 4 positive edges of the clock!** In other words, the instruction is fetched from memory on the very **1st** positive edges, and is in the pipeline for **three** full cycles before the result is written back.

When designing your processor, try to keep track of your critical path. You **will not** be able to meet the timing requirement if you try to do everything in 1 cycle! When considering how to break down your pipeline, consider any potential data or control hazards that may arise and how you will resolve them. **You are not allowed to resolve hazards by stalling your processor!**

Since you will be graded on the logic utilization of your processor, you will want to review the Virtex-5 datasheet to see how your code will map into actual hardware. You may also wish to familiarize yourself with **Synplify Pro** to take advantage of the various optimization features and options that it offers.

Figure 2 The 3-stage pipeline


clock
I
X
M
The PC is updated and instruction fetch begins.
DMem read or write begins.
RegFile write begins. New instruction starts.

5 I/O Interface and Memory System

The processor will be using Memory-mapped I/O to talk to all other components in your project. Aside from a global `Reset` signal, the memory ports will serve as the only way to transfer information to and from the processor. The port specification for the processor is given in Table 3. This interface is identical to the processor interface presented to you in Lab 5 (although a few wires were renamed for clarity).

Table 3 Port specification for the processor

Signal	Width	Dir	Description
`Clock`	1	I	The Clock input to the processor
`Reset`	1	I	The global Reset signal
`MemoryAddress`	32	O	The address corresponding to a memory or I/O transaction
`MemoryReadData`	32	I	The data that was read from memory or an I/O device
`MemoryRead`	1	O	Asserted by the processor on a memory or I/O read transaction
`MemoryWriteData`	32	O	The data to be written to memory or an I/O device
`MemoryWrite`	1	O	Asserted by the processor on a memory or I/O write transaction

Note: we may change the sizes of data and instruction memory in the coming weeks, so design your memory module in in a way that permits the size to be changed.

I/O devices, as well as Instruction and Data memories, are memory-mapped onto the address space. Table 4 shows how the address space is divided. Note that this is only a partial address map right now. We will be adding more components later in the semester.

Keep in mind that the memory system is **byte-addressed**. Accesses to memory or I/O should also be **word-aligned**. Also, note that several sections of the address space are **write-only** or **read-only**. If the processor attempts to write to a read-only field, then the write has no effect. If the processor attempts to read from a write-only field, then the resulting data could be anything.

Take special note that **Instruction Memory** is marked as **write-only** in the address space. This just means that the processor cannot use its load/store memory ports to read from these addresses. **Note also that both Instruction and Data memories will implement synchronous read.** The instruction fetch stage of your processor pipeline, however, will need a dedicated read port to the **Instruction Memory** in order to fetch one instruction per cycle. **The instruction fetch stage should be the only part of your processor that reads from the Instruction Memory**.

Table 4 Map of the MIPS150 partial address space

Addresses	Read/Write	Purpose
`0x00400000 - 0x00407ffc`	W	Instruction Memory (User code)
`0x7ffff000 - 0x7ffffffc`	R/W	Data Memory (Stack)
`0x10010000 - 0x10017ffc`	R/W	Data Memory (Heap)
`0xffff0000 - 0xffff0000`	R	Serial Interface (ControlInReg)
`0xffff0004 - 0xffff0004`	R	Serial Interface (DataInReg)
`0xffff0008 - 0xffff0008`	R	Serial Interface (ControlOutReg)
`0xffff000c - 0xffff000c`	W	Serial Interface (DataOutReg)

6 Restrictions

Although the design and implementation of the processor is meant to be open-ended, the following is the small set of restrictions you are expected to obey.

1. Your design must be written in Verilog.
2. Your design must adhere to the standard MIPS ISA for the subset of instructions you are implementing.
3. Your design must not exceed the amount of resources available on the Virtex-5 LX110T. For example, your project should not be using more Block RAMs than the number available on the board.
4. Clocks per Instruction (CPI) for your processor must be equal to 1. In other words, **no stalling!**.
5. Your processor must be **able to run at 50-100 MHz (this number will be fixed in the next 2 weeks!)** without any setup or hold time violations.
6. Your processor must have 3 pipeline stages. For information on what is considered a 3-stage pipeline, refer back to Figure 2.

Rev.	Name	Date	Description
I	Chris Fletcher	4/7/2010	Added the stack to the memory map.
H	Chris Fletcher	3/29/2010	Moved data memory (heap) to `0x10010000`.
G	Ilia Lebedev	3/2/2010	Added clarification on what “3 stage pipeline” mans. Clarified that instruction and data memories read synchronously. Fixed RTL bugs in the ISA table.
F	Ilia Lebedev & Chris Fletcher & John Wawrzynek	2/27/2010	Adopted for Spring 2010.
A-E	...	Spring 2009	Spring 2009 EECS150 Staff.