Assignment 2

Group 4

February 28, 2012

1 Instruction set

This group has chosen to implement the core instruction of the MIPS processor. This core instruction set consists of the following instructions:

NAME	MNEMONIC	FORMAT	OPERATION
Add	add	R	R[rd] = R[rs] + R[rt]
Add Immediate	addi	I	R[rt] = R[rs] + SignExtImm
Add Imm. Unsigned	addiu	I	R[rt] = R[rs] + SignExtImm
Add Unsigned	addu	R	R[rd] = R[rs] + R[rt]
And	and	R	R[rd] = R[rs] & R[rt]
And Immediate	andi	I	R[rt] = R[rs] & ZeroExtImm
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr
Jump	j	J	PC=JumpAddr
Jump And Link	jal	J	R[31]=PC+8;PC=JumpAddr
Jump Register	jr	R	PC=R[rs]
Load Byte Unsigned	lbu	I	R[rt]=24'b0,M[R[rs]+SignExtImm](7:0)
Load Halfword Unsigned	lhu	I	R[rt]=16'b0,M[R[rs]+SignExtImm](15:0)
Load Linked	ll	I	R[rt] = M[R[rs]+SignExtImm]
Load Upper Imm.	lui	I	R[rt] = imm, 16'b0
Load Word	lw	I	R[rt] = M[R[rs]+SignExtImm]
Nor	nor	R	R[rd] = (R[rs] — R[rt])
Or	or	R	R[rd] = R[rs] — R[rt]
Or Immediate	ori	I	R[rt] = R[rs] — ZeroExtImm
Set Less Than	slt	R	R[rd] = (R[rs] ¡ R[rt]) ? 1 : 0
Set Less Than Imm.	slti	I	R[rt] = (R[rs] ¡ SignExtImm)? 1 : 0
Set Less Than Imm. Unsigned	sltiu	I	R[rt] = (R[rs] ¡ SignExtImm) ?1:0
Set Less Than Unsig.	sltu	R	R[rd] = (R[rs] ¡ R[rt]) ? 1 : 0
Shift Left Logical	sll	R	R[rd] = R[rt] ¡¡ shamt
Shift Right Logical	srl	R	R[rd] = R[rt] ¿¿ shamt
Store Byte	sb	I	M[R[rs]+SignExtImm](7:0) = R[rt](7:0)
Store Conditional	sc	I	M[R[rs]+SignExtImm] = R[rt]; R[rt] = (at
Store Halfword	sh	I	M[R[rs]+SignExtImm](15:0) = R[rt](15:0)
Store Word	sw	I	M[R[rs]+SignExtImm] = R[rt]
Subtract	sub	R	R[rd] = R[rs] - R[rt]
Subtract Unsigned	subu	R	R[rd] = R[rs] - R[rt]

TODO: referrence needed (taken from MIPS-Greencard)

Since we are working with the basic MIPS instructions, there are already numerous sources describing the instructions in great detail, so instead of further describing the instructions, we will expand upon our motivation for choosing the MIPS core instruction set, and how we hope to expand on it, once it has been successfully implemented.

1.1 Motivation

The MIPS instruction set is well tested, and has a lot of advantages. The fixed length 32-bit instruction words allow for smarter hardware implementation, especially since the structure of the instruction words is fixed, so that instructions can be passed on through pipelined stages and be decoded along the way, instead of having to be decoded all at once.

By using the full core instruction set, we have the added advantage of being able to compile real programs into MIPS assembler code, which can be run directly on our processor. Had we instead chosen to work on a modified version of the MIPS instruction set or a smaller subset, a compiler would have had to be written from scratch, or any program that was to run on the processor would have to be written in assembler. Both these solutions would involve a lot of work, which would not be directly relevant for the purpose of this course.

1.2 Future work

Once a functioning MIPS processor is up and running, we hope to be able to expand the processors instruction set, and are currently considering a number of options. One obvious expansion could be the inclusion of floating point instruction, which would, of course, require an implementation of a floating point co-processor. This has the added advantage, that we would still be able to use existing compilers to generate our assembler code.

An alternative could be to expand the instruction set with new instructions (e.g. more elaborate one-cycle branch instructions such as 'branch if less than' or others). These would however require some sort of custom assembler code to be generated, but we will cross that bridge when (and if) we get to it.

Yet another possibility would be to work on parts of the processor design not directly related to the instruction set. This could for instance be to work on the memory access and hierarchy, and / or getting the FPGA to output simple graphics from the MIPS over the VGA output.