

Project Proposal

1. Team Name and Members

Team name: Ohio

Team leader: Sam Fischbach

Team members: N/A, N/A

2. Design Overview

The design of the system will generally follow a MIPS architecture, utilizing a word size of 16 bits. Instructions will be a subset of MIPS32 core instructions (in 16 bits) including arithmetic, logical, and memory operations (specified in more detail below.) Instructions will be of three types – R, I, or J – and will comprise a total of 16 operations.

Operations will clock-trigger on the rising edge.

The register file will contain only 8 16-bit registers, which is a (just) capable amount.

0-1 t0-t1

2-5 v0-v3

6-7 a0-a1

This arrangement is inconvenient – especially the lack of a zero register – but workable by using some extra instructions.

2.1. Instruction format

R-Format:

Opcode	rs	rt	rd	Func
4 bits	3 bits	3 bits	3 bits	3 bits

I-Format:

Opcode	rs	rt	immediate
4 bits	3 bits	3 bits	6 bits

J- Format:

Opcode	address
4 bits	12 bits

2.2. Instructions

Name	Mne-	Operation	Opcode	Func	Format
Add	add	add \$s1, \$s2, \$s3; $\$s1 = \$s2 + s3$	0000	000	R
Subtract	sub	sub \$s1, \$s2, \$s3; $\$s1 = \$s2 - \$s3$	0000	001	R
AND	and	and \$s1, \$s2, \$s3; $\$s1 = \$s2 \& \$s3$	0000	010	R
OR	or	or \$s1, \$s2, \$s3; $\$s1 = \$s2 \mid \$s3$	0000	011	R
NOR	nor	nor \$s1,\$s2,\$s3; $\$s1 = \sim(\$s2 \mid \$s3)$	0000	100	R
XOR (Exclusive Or)	xor	Xor \$s1,\$s2,\$s3; $\$s1 = \$s2 \oplus \$s3$	0000	101	R
Set on Less Than	slt	slt \$s1,\$s2,\$s3; $\$s1 = 1:\$s2 < \$s3 \mid 0$ else	0000	110	R

Add Immediate	addi	addi \$s1,\$s2,57; $\$s1 = \$s2 + 57$	0010	...	I
Load Word	lw	lw \$s1,off6(\$s2); $\$s1 = \text{MEM16}(\$s1 + \text{off6})$	0011	...	I
Store Word	sw	sw \$s1,off6(\$s2); $\text{MEM16}(\$s1 + \text{off6}) = \$s1$	0100	...	I
Branch on Equal	beq	beq \$s1, \$s2, off6; If $\$s1 = \$s2$, PC += off6	0101	...	I
Set on Less Than Immediate	slti	slti \$s1, \$s2, 5; $\$s1 = 1:\$s2 < 5 \mid 0$ else	0110	...	I
NOP	nop	No operation	1111	...	I
Shift Left Logical	sll	sll \$s1, \$s2, 2; $\$s1 = \$s2 \ll \text{shift2}$	1001	...	I
Shift Right Logical	srl	srl \$s1, \$s2, 4; $\$s1 = \$s2 \gg \text{shift4}$	1010	...	I

Jump	j	j off12	1000	...	J
------	---	---------	------	-----	---

2.3. Assembly language and machine code for the test program (Pseudocode)

Pseudocode for the test program from project_instructions.pdf:

```
$v0 = 0040hex; // you can redefine $v0-3, $t0, and $a0-1 with
$v1 = 1010hex; // your register numbers such as $1, $2, etc.
$v2 = 000Fhex;
$v3 = 00F0hex;
$t0 = 0000hex;
$a0 = 0010hex;
$a1 = 0005hex;
while ($a1 > 0) do {
    $a1 = $a1 - 1;
    $t0 = Mem[$a0];
    if ($t0 > 0100hex) then {
        $v0 = $v0 ÷ 8;
        $v1 = $v1 | $v0; //or
        Mem[$a0] = FF00hex;
    }
    else {
        $v2 = $v2 × 4;
        $v3 = $v3 ⊕ $v2; //xor
        Mem[$a0] = 00FFhex;
    }
    $a0 = $a0 + 2;
}
return;
```

Assembly / Machine Code:

Notes:

Initial values of memory and registers are preloaded. Code below begins with program loop.

This design lacks a zero register, which makes assigning register values more difficult. We split it into two instructions: `slt a,a,a` sets register a to zero, then `addi a,a,const` places a value into the zeroed register. (This could of course be turned into a pseudo-instruction on this architecture, but that's outside the scope of this project.)

Immediates only have a 6-bit constant value. Since we typically operate on powers of two, only 4 bits will be useful, but we can shift and add. Inefficient in terms of instruction count.

Address	Assembly	Comment	Machine Code	T
0x0000	Loop:	#Begin loop test		
0x0000	slt \$t0, \$t0, \$t0	#Reset t0 = 0	0x0006: 0000 000 000 000 110	R
0x0002	addi \$t0, \$t0, 1	#Set t0 = 1	0x2001: 0010 000 000 000001	I
0x0004	slt \$t1, \$a1, \$t0	#t1=0 while a > 0	0x0E0E: 0000 111 000 001 110	R
0x0006	beq \$t0, \$t1, End*	#When t1=1, jump to end	0x521E: 0101 001 000 011110	I
0x0008	NOP	#Begin the loop logic	0xF000: 1111 0000 0000 0000	
0x000A	addi \$a1, \$a1, -1	#Decrement a1	0x2FFF: 0010 111 111 111111	I
0x000C	lw \$t0, 0[\$a0]	#Load mem at a0 into t0	0x3C00: 0011 110 000 000000	I
		#If statement begins		
0x000E	slt \$t1,\$t1,\$t1	#Set t1 = 0	0x024E: 0000 001 001 001 110	R
0x0010	addi \$t1, \$t1, 0x01	#t1 = 0001h	0x2241: 0010 001 001 000001	I
0x0012	sll \$t1, \$t1, 8	#t1 = 0100h	0x9248: 1001 001 001 001000	I
0x0014	slt \$t1, \$t1, \$t0	#t1=1 while t0 > 0100h	0x020E: 0000 001 000 001 110	R
0x0016	slt \$t0, \$t0, \$t0	#OK since t0 already used	0x0006: 0000 000 000 000 110	R
0x0018	beq \$t1, \$t0, Else**	#Go to Else when t0<0100h	0x504B: 0101 000 001 001011	I
0x001A	NOP	# If “true” part	0xF000: 1111 0000 0000 0000	
0x001C	srl \$v0,\$v0,3	#v0/8, unsigned	0xA483: 1010 0100 1000 0011	I
0x001E	or \$v1, \$v1, \$v0	# v1 = v1 v0	0x069B: 0000 011 010 011 011	R
0x0020	slt \$t0,\$t0,\$t0	#Zero out t0	0x0006: 0000 000 000 000 110	R
0x0022	addi \$t0,\$t0, 0xF	#t0 = 000F h	0x200F: 0010 000 000 001111	I
0x0024	sll \$t0,\$t0,4	#t0 = 00F0 h	0x9004: 1001 000 000 000100	I
0x0026	addi \$t0,\$t0, 0xF	#t0 = 00FF h	0x200F: 0010 000 000 001111	I
0x0028	sll \$t0,\$t0,8	#t0 = FF00 h	0x9008: 1001 000 000 001000	I
0x002A	sw \$t0, 0[\$a0]	#Store t0 at [a0]	0x4C00: 0100 110 000 000000	I
0x002C	j Endif***	#Jump over the Else part	0x803E: 1000 0000 0011 1110	J
0x002E	NOP			
0x0030	Else:	#Else Part		
0x0030	sll \$v2, \$v2, 2	#v2 *= 4	0x9902: 1001 100 100 000010	I
0x0032	xor \$v3, \$v3, \$v2	# v3 = v3 ⊕ v2	0x0B2D: 0000 101 100 101 101	R
0x0034	slt \$t0,\$t0,\$t0	#Zero out t0	0x0006: 0000 000 000 000 110	R
0x0036	addi \$t0,\$t0,0xF	#t0 = 000F h	0x200F: 0010 000 000 001111	I
0x0038	sll \$t0,\$t0,4	#t0 = 00F0 h	0x9004: 1001 000 000 000100	I
0x003A	addi \$t0,\$t0,0xF	#t0 = 00FF h	0x200F: 0010 0000 0000 1111	I
0x003C	sw \$t0, 0[\$a0]	#Store t0 at [a0]	0x4C00: 0100 110 000 000000	I

0x003E	Endif:	#If Statement done		
0x003E	addi \$a0,\$a0,2	#Inc a0 by 2	0x2D82: 0010 110 110 000010	I
0x0040	j Loop****	#Start over	0x8000: 1000 000000000000	J
0x0042	NOP	(nop)	0xF000: 1111 0000 0000 0000	
0x0044	End:	(nop)	0xF000: 1111 0000 0000 0000	

Addresses – Loop: 0x0000, Else: 0x002E, Endif: 0x003C, End: 0x0040

Relative addressing for beq1(06) and beq2(18), pseudodirect for j(3E)

* 44 – (06+2) = 3Csr1 = 01 1110b *** 0x003E = 0000 0011 1110

30 – (18 +2) = 16sr1 = 001 011b ** 0x0000 = 0000 0000 0000

3. Tasks and Schedule

- Week 1:** Produced proposal, outlining basis for MIPS multi-cycle pipeline implementation and ISA of design.
Produced initial pass of assembly instructions for test program and considered pipeline design and layout. Assembled table of preliminary control values.
- Week 2:** Placed Jump and Branch computation in ID stage, decision in EX stage.
Rearranged instruction OpCodes for greater consistency and streamlining.
Revised instruction code, inserting NOP after branch and jump operations.
Updated relative and direct target addresses and byte code.
Produced new table of control settings for revised instructions and pipeline.
Laid out bare structure for simulation.
- Week 3:** Produced bare-bones simulation of single-cycle MIPS pipeline that read in initial values and executed code.
Began work on multi-cycle implementation and GUI front-end (for both greater ease of debugging and end-user.)
Corrected number of code and translation errors; probably should have written a 'lite' compiler to recompute binary and byte code from instructions, rather than doing it by hand.
- Week 4:** Continue work on simulator. Perfect hazard detection and forwarding mechanisms.
Continue debugging. Clean up code, draft final report documentation.