8 bit Processor
By: Team MIPS Don't Lie

Instruction Set

Name / Mnemonic	Opcode (Binary)	Operation
Accumulate | acm	000	ACC = R[rt]
Accumulate Immediate | acmi	001	ACC =SignExt(immediate)
Add | add	010	R[rt] = R[rt] + ACC
NAND | nand	011	R[rt] = ~ (R[rt] & ACC)

Branch Not Zero and Link | bnzl	100	If (ACC != 0)
		PC = R[rt]
		else
		PC = PC
		R[rt] = PC

Set Less than | slt	101	R[rt] = (ACC < R[rt]) ? 1 : 0
Store Word | sw	110	M[ACC] = R[rt]
Load Word | lw	111	R[rt] = M[ACC]

*SignExt(X) => Sign Extension of X
*M[X] => Value in memory at address X
*R[X] => Value at register addressed by X

Basic Instruction Formats

opcode		immediate / rt	
7	5	4	0

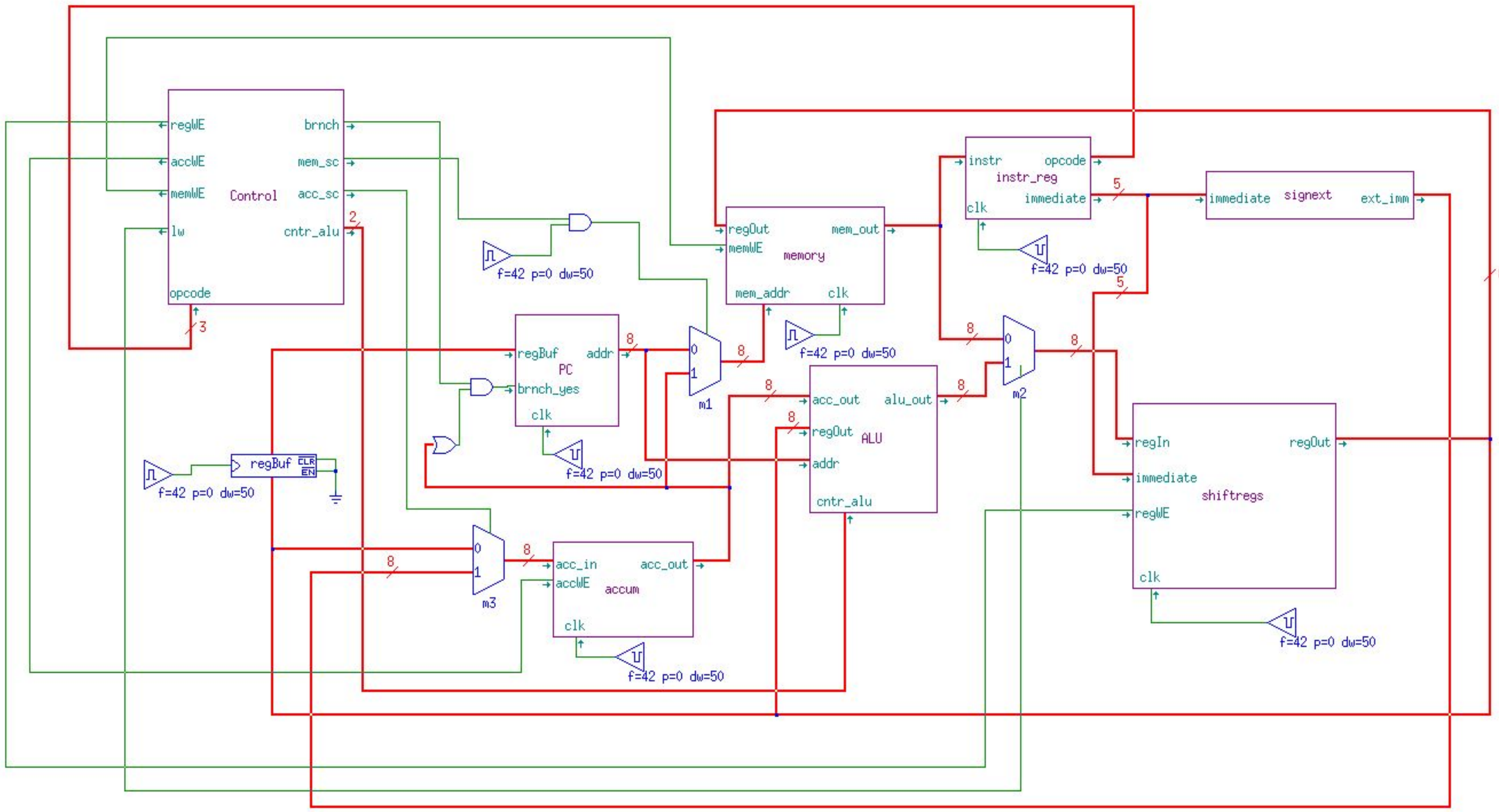
Pseudo Instructions

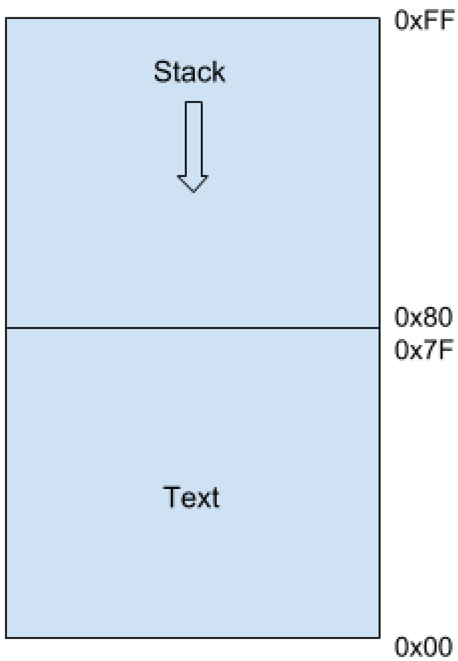
bnzl Label $REG	Branch if $REG != 0, to line with Label and link PC to $ra
bnzl Label 3-bit-Val	Branch if 3-bit-Val != 0, to line with Label and link PC to $ra

Registers

Name	Number	Use	Preservation Convention
$zero	0	0	Yes (always 0)
$v0-$v3	1-4	Return Values	No
$a0-$a3	5-8	Arguments	No
$s0-$s7	9-16	Saved Temporaries	Yes
$t0-$t10	17-27	Temporaries	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$ra	30	Return Address	Yes
$HR	31	Halt Register	---
Non Addressable:			
PC	Program Counter	Point to Address of Current Instruction	---
ACC	Accumulator Register	Stores a value for subsequent use	---

Design + Explanation





Memory Hierarchy

Memory

- The instructions are loaded from the object file, mem.bin, into a 256 by 8 memory array with the instruction memory loaded starting at address 0x00 ending at max 0x7F.
- The stack starts at address 0xFF and ends at 0x80.
- Address is selected by MUX between PC and Accumulator
- Memory is written to (if enabled) on negative edge clock

PC

- PC increments by 0x01 on positive edge clock
- PC changes to the Branch Address value (stored in the register file's output buffer register) when brnch_yes is enabled
    - brnch_yes is enabled when brnch control value is high and at least one bit of the accumulator is not zero

Register File (shiftregs)

- All registers in Register File are initialized to 0x00 except for the stack pointer($sp) which is initialized to top of Stack (0xFF)
- The addressed register is written to (if enabled) on negative edge clock
- The Register file is addressed by the immediate portion of the instruction (see instruction format above)
- There are 32 registers because that is the maximum that can be addressed with the 5 bits chosen as the immediate value

Instruction Register

- On positive edge clock, reads instruction from memory and holds its value until next clock cycle.
- Acts as buffer for instruction so that when other parts of memory need to be accessed, the cpu won't lose track of current instruction.
    - This is only necessary because the cpu is an implementation of Von-Neumann architecture meaning instruction and data memory exist on the same memory array.

Accumulator

- non-addressable register
- Input is selected by MUX between the output of the ALU and the Register File
- Written to on negative edge clock when acm or acmi instructions are active

ALU

- A 2-bit control signal determines operation: ‘add’, ‘nand’, ‘branch address vs. current address’, and ‘less than’.
- The first input is always the value held in the accumulator. The second is from the register file, and the third input is the current address held in PC.

Control Unit

- A Moore type finite state machine to control all the components based on the opcode (first 3 bits of instruction)

Truth Table for Control Unit

	cntr_alu	regWE	memWE	brnch	lw	accWE	selAccIn	selMemIn
ACM	XX	0	0	0	X	1	0	0
ACMI	XX	0	0	0	X	1	1	0
ADD	00	1	0	0	0	0	X	0
NAND	01	1	0	0	0	0	X	0
BNZL	10	1	0	1	0	0	X	0
SLT	11	1	0	0	0	0	X	0
SW	XX	0	1	0	X	0	X	1
LW	XX	1	0	0	1	0	X	1

Assembler Syntax

```
acmi 6            //Instruction Immediate
add $a0           //Instruction Register

acm $v0 MUL:  //Instruction Register Label   Label points to the address of line it is on

bnzl MUL $t1   //bnzl pseudo instruction. This one will branch to line with “acm %v0 MUL:” if $t1
               //doesn’t contain value of 0

bnzl MUL 001  //bnzl pseudo instruction. This one will branch to line with “acm %v0 MUL:” if 001
              //is not 0 which it isn’t. Note the leading zeroes are necessary for assembler to
              run
              //without error

acmi 1
add $HR        //add a value to the halt register to terminate program.
```

Assembling and Linking

The assembling and linking is performed all by the assembler.py python program. It can be run by either typing:

python2 assembler.py < “application.asm” > mem.bin

Or by using the makefile and setting file parameter to the assembly file:

make file=application.asm

Sample Programs

1. mult.asm (multiply two integers)
2. fib.asm (find the first n number of specified fibonacci numbers)
3. fact.asm (factorial of a number)

Timing

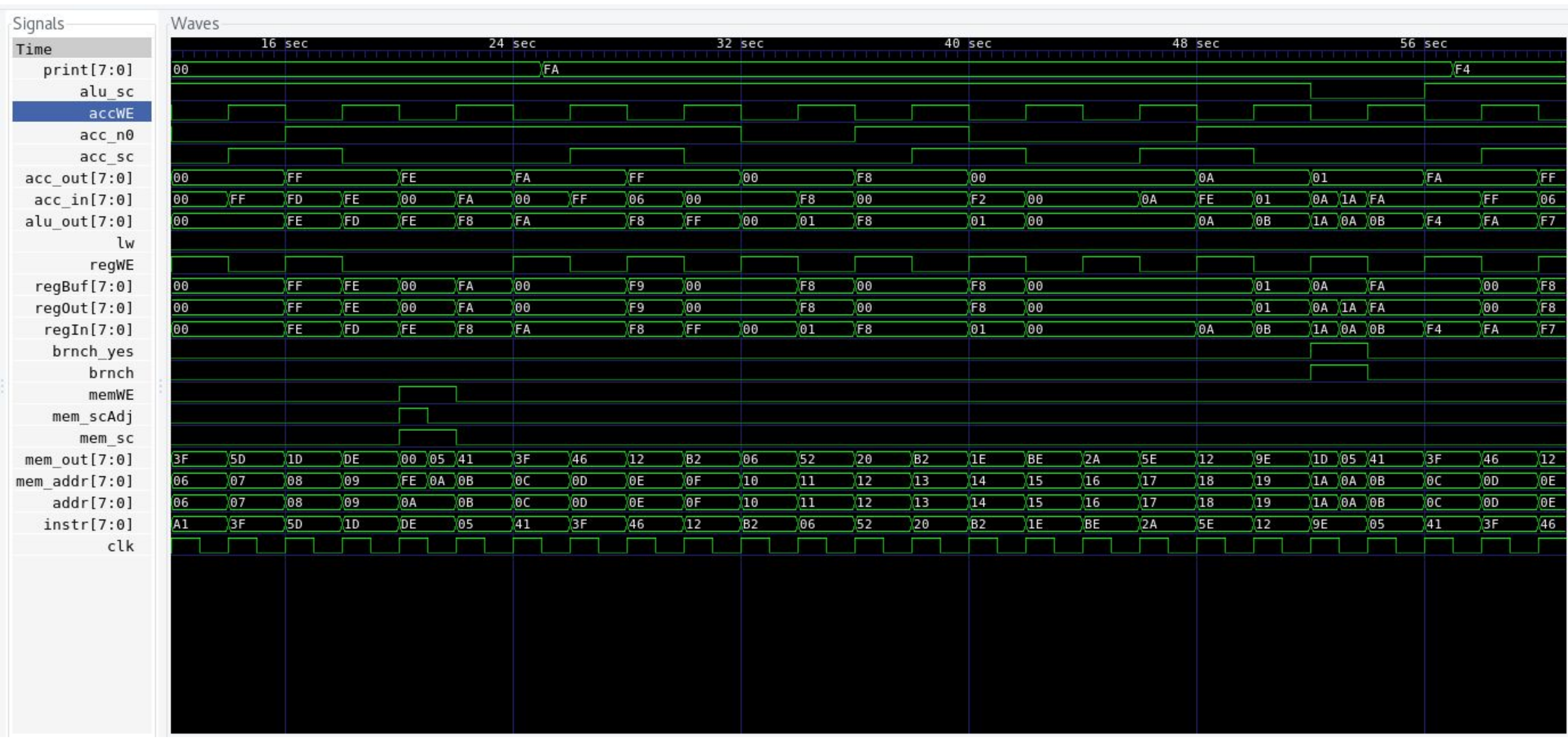
By using the makefile and gtkwave run the following to view a full timing diagram:

    make time

Example Diagram: (for mult.asm)

Time 19-21: store word - so memory address is changing twice in clock cycle.

Time 52-54: branching - so pc address (addr) is changing twice in clock cycle.



Overall Pro's and Con's

- ➔ Although only 8 instructions the 8 instructions above are enough to perform both leaf, nested, and recursive application programs
- ➔ Because only 8 instructions we need only 3 bits for the opcode. Thus we have a rather large immediate value range for just an 8 bit computer and we still have the full 32 registers that MIPS has.
- ➔ Although we have only 256 bytes of main memory, memory doesn't need to be used as often because there are a lot of registers available.
- ➔ Just a one line instruction format means a small clock cycle which is important for a single cycle processor.