Mini-MIPS 8-bit Processor

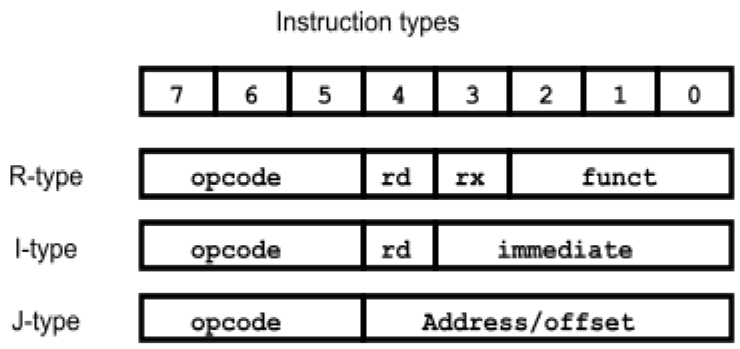
OVERVIEW

An 8-bit MIPS-like single-cycle CPU.

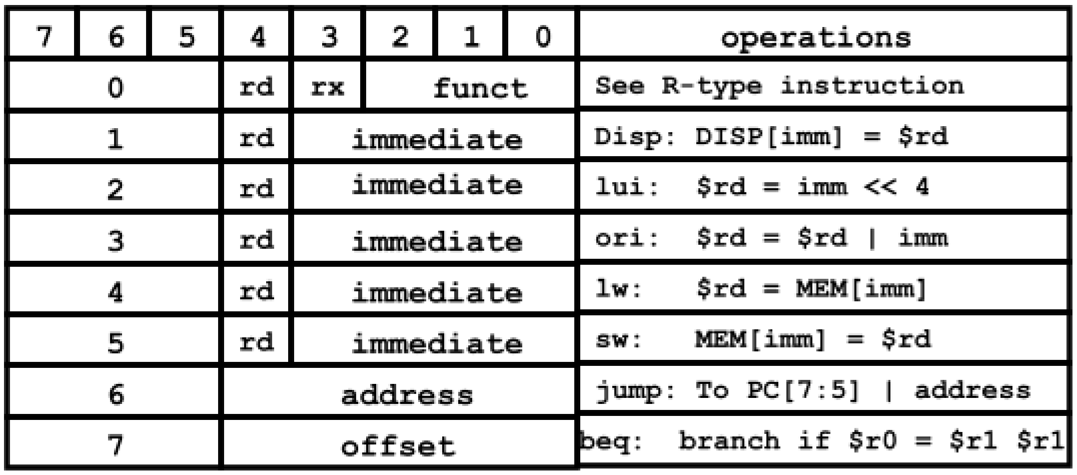
ISA

This is the Instruction set for the simple 8-bit processor. There is only two 8-bit registers (**$r0** and **$r1**). There is an 8-bit **PC**.

Mini-MIPS has separate instruction memory and data memory. Each has maximum capacity of **256 bytes**.



INSTRUCTION FORMATS



7	6	5	4	3	2	1	0	operations
0			rd	rx	funct			See R-type instruction
1			rd	immediate				Disp: DISP[imm] = $rd
2			rd	immediate				lui: $rd = imm << 4
3			rd	immediate				ori: $rd = $rd | imm
4			rd	immediate				lw: $rd = MEM[imm]
5			rd	immediate				sw: MEM[imm] = $rd
6			address					jump: To PC[7:5] | address
7			offset					beq: branch if $r0 = $r1 $r1

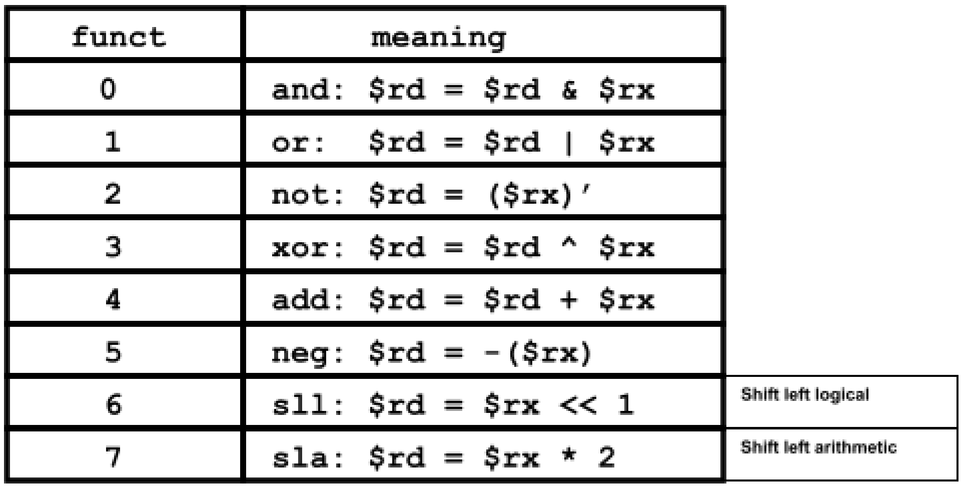
Notation:

<<	# left shift with zero into low order bits
|	# concatenation
address	# unsigned number
Offset	# signed offset

immediate field
All immediate fields are treated as unsigned numbers and they are zero extended(Upper 3 bits will be all zeroes)

R-TYPE INSTRUCTIONS

R-type instructions



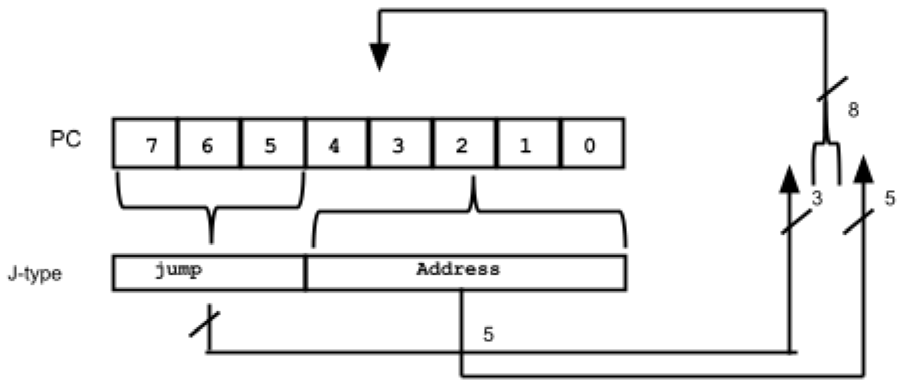
funct	meaning	
0	and: $rd = $rd & $rx	
1	or: $rd = $rd | $rx	
2	not: $rd = ($rx)′	
3	xor: $rd = $rd ^ $rx	
4	add: $rd = $rd + $rx	
5	neg: $rd = -($rx)	
6	sll: $rd = $rx << 1	Shift left logical
7	sla: $rd = $rx * 2	Shift left arithmetic

I-TYPE INSTRUCTIONS

lui (load upper immediate):
Loads an **immediate** value into the upper bits of a register.
sw (store word):
Stores the value from a register into **memory** at an address **offset**.
lw (load word):
Loads a value from memory into a register using an **immediate** offset.
ori (bitwise OR immediate):
Performs a **bitwise OR** between a register and an immediate value.
disp (display):
See the section on the 7-segment display, below.

J-TYPE INSTRUCTIONS

jump instruction
jump instruction has 5-bit address. The address(5-bit) will be concatenated with the upper 3 bits of current value of PC.



beq instruction

7	offset

`beq:  branch if $r0 = $r1`

The offset is a **signed number**, therefore the branch address is calculated as follows. The registers are implied and do not appear in the instruction itself.

```
if $r0 == $r1
    PC = (PC + 1) + offset(sign extended)
else
    PC = PC + 1
```

ROM & RAM MODULES

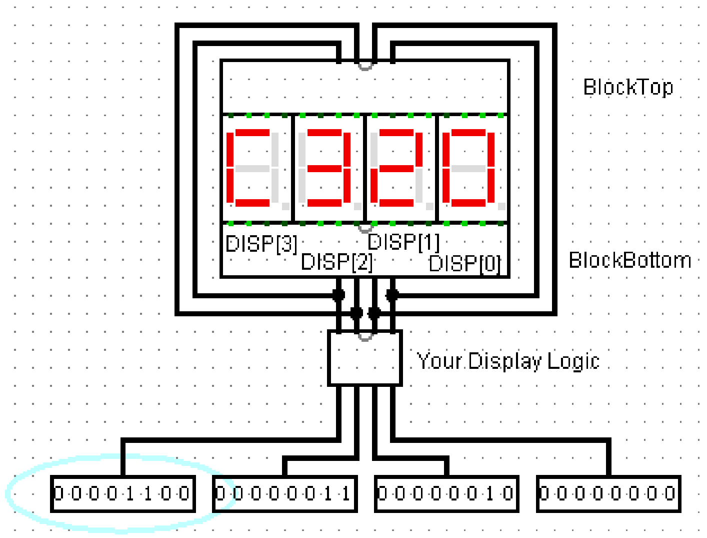
Instruction memory uses the ROM module to load your program. Data memory uses the RAM module..

Note: The ROM & RAM appear to be organized in 32-bit words but they are still "byte-addressable memory". Also, remember that our word size is only one byte.

INPUT & OUTPUT

Input
Input will be a program that you will write and assemble(by hand) and load into the Instruction memory module. Use a ROM module to store instructions.

Output
Output will be displayed on an array(4 of them) of 7-segment display module.

7-Segment Display


The Disp instruction displays the content of a register to the *imm***th** 7-segment display unit. In the above picture, there are four 7-segment display units.

For example, DISP $r1 [2] # $r1's content will be displayed on 3rd unit—in this case, the imm= 2 (decimal) 00010 which points to the 3rd unit from right.

If the value of a register $r1 is 00000011(0x03), the display unit will display "3" as above.

TESTING

Here is a sample program that you can run as a basic test. The program loads 15(in decimal=F in hex) into register $r0 and display. We do the same for $r1. Then we branch by *beq -1* (it goes back to itself! In other words, “halt”). The result is to display FF on the 7-segment display and stops the program. It is a Display-and-Halt program.

```
lui     $r0, 0          # load upper 4 bits with all zeroes
ori     $r0, 15         # or with 15(F in hex)
disp    $r0, 0          # display the content of $r0 on 0th display unit
lui     $r1, 0          # load upper 4 bits with all zeroes
ori     $r1, 15         # or with 15(F in hex)
disp    $r1, 1          # display the content of $r1 on 1th display unit
beq     -1              # Halt
```