

Problem 1)

ISA Description and Opcodes

|

ISA

Instruction	OPcode	Description	Format
-----	-----	-----	-----
`add`	`00`	Adds rA to rB	`00 rA rB`
`sub`	`01`	Subtracts rA from rB	`01 rA rB`
`mul`	`02`	Multiplies rB by rA	`02 rA rB`
`div`	`03`	Divides rB by rA	`03 rA rB`
`mrmov`	`40`	Moves D(rB) to rA	`40 rA rB D` (1 byte)
`rmmov`	`50`	Moves rA to D(rB)	`50 rA rB D` (1 byte)
`rrmov`	`60`	Moves rB to rA	`60 rA rB`
`irmov`	`70`	Moves V to rA	`70 rA F V` (1 byte)
`jmp`	`90`	Unconditional jump to Dest	`90 FF Dest` (1 byte)
`jle`	`91`	Jump <= to Dest	`91 FF Dest` (1 byte)
`jl`	`92`	Jump < to Dest	`92 FF Dest` (1 byte)
`je`	`93`	Jump == to Dest	`93 FF Dest` (1 byte)
`jne`	`94`	Jump != to Dest	`94 FF Dest` (1 byte)
`jg`	`95`	Jump > to Dest	`95 FF Dest` (1 byte)
`jge`	`96`	Jump <= to Dest	`96 FF Dest` (1 byte)
`halt`	`A0`	Stops execution	`A0`
Maybe			
`push`	`B0`	Push rA to stack	`B0 rA 0 01`
`pop`	`C0`	Pop rA from stack	`C0 rA 0 01`

Registers

Register	Code
-----	----
%eax	0
%ebx	1
%ecx	2
%edx	3
%rsp	4
%rbp	5

Flags

Flag	Code
-----	----
No Flag	00
Neg Flag	01
Zero Flag	10

Problem 2)

The assumptions for Matrix A, B, and C are that they are all 2x2 matrices. The corresponding values from the first value in matrix A and B should be added to produce the first value in matrix C. Continue this for the other 3 values. This produces matrix C. Eax holds the value of matrix A and is added to ebx which holds the matrix B value. The resulting ebx value is pushed for the corresponding matrix C value.

Main:

```
*****Setting up base and stack pointer, filling stack
#           with matrix values *****
irmovl $0xFC, %esp
irmovl $0xFE, %ebp
irmovl $0x05, %eax
pushl %eax
irmovl $0x02, %eax
pushl %eax
irmovl $0x04, %eax
pushl %eax
irmovl $0x03, %eax
pushl %eax
irmovl $0x01, %eax
pushl %eax
irmovl $0x07, %eax
pushl %eax
irmovl $0x09, %eax
pushl %eax
irmovl $0x02, %eax
pushl %eax

#Add first element and push to stack
irmovl $0xF8, %ecx
mrmovl (%ecx), %eax
irmovl $0xE8, %ecx
mrmovl (%ecx), %ebx
addl %eax, %ebx
pushl %ebx

#Add second element and push to stack
irmovl $0xF4, %ecx
mrmovl (%ecx), %eax
irmovl $0xE4, %ecx
mrmovl (%ecx), %ebx
addl %eax, %ebx
pushl %ebx

#Add third element and push to stack
irmovl $0xF0, %ecx
mrmovl (%ecx), %eax
irmovl $0xE0, %ecx
mrmovl (%ecx), %ebx
addl %eax, %ebx
pushl %ebx

#Add fourth element and push to stack
irmovl $0xEC, %ecx
mrmovl (%ecx), %eax
irmovl $0xDC, %ecx
mrmovl (%ecx), %ebx
addl %eax, %ebx
pushl %ebx
```

```

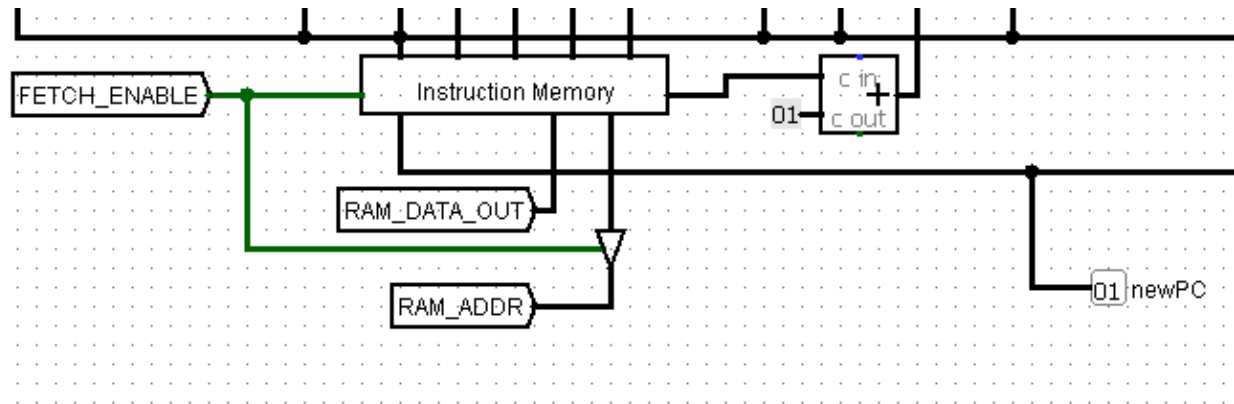
[simonyao]@compute ~/CSCE312/Lab5/sim/y86-code> (18:00:51 04/23/19)
:: ../misc/yis finalproject.yo
Stopped in 43 steps at PC = 0xbc.  Status 'HLT', CC Z=0 S=0 O=0
Changes to registers:
%eax: 0x00000000      0x00000003
%ecx: 0x00000000      0x000000dc
%ebx: 0x00000000      0x00000005
%esp: 0x00000000      0x000000cc
%ebp: 0x00000000      0x000000fe

Changes to memory:
0x00cc: 0x00000000      0x00000005
0x00d0: 0x00000000      0x0000000d
0x00d4: 0x00000000      0x00000009
0x00d8: 0x00000000      0x00000006
0x00dc: 0x00000000      0x00000002
0x00e0: 0x00000000      0x00000009
0x00e4: 0x00000000      0x00000007
0x00e8: 0x00000000      0x00000001
0x00ec: 0x00000000      0x00000003
0x00f0: 0x00000000      0x00000004
0x00f4: 0x00000000      0x00000002
0x00f8: 0x00000000      0x00000005

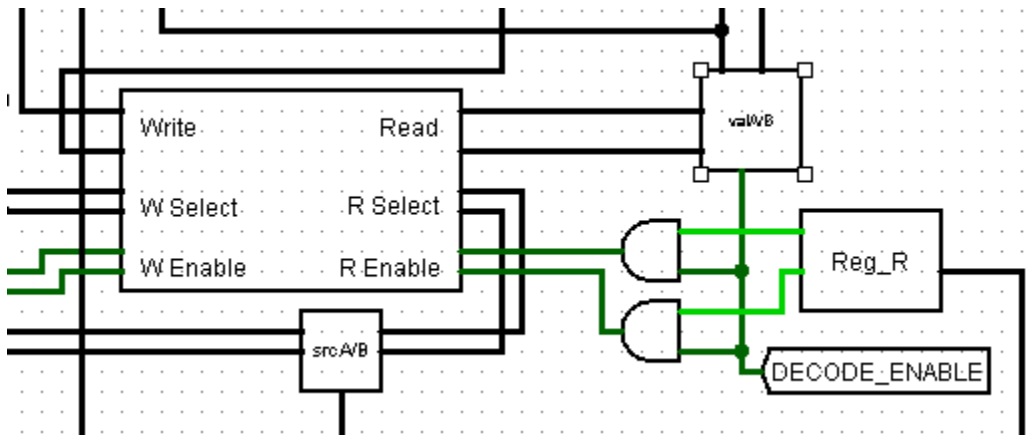
```

Our ISA program:

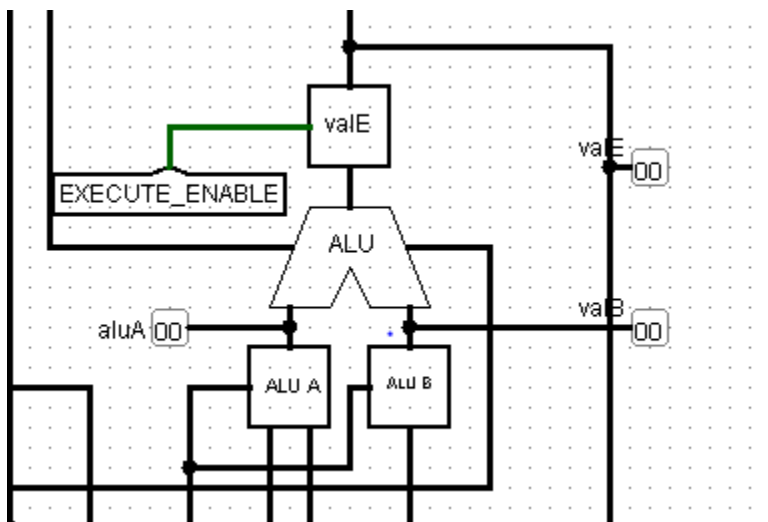
irmov	FE -> %ebp	70 5 0 FE
irmov	F7 -> %esp	70 4 0 F7
irmov	FE -> %ecx	70 2 0 FE
rrmov	%ecx -> %eax	40 0 2 00
irmov	FA -> %ecx	70 2 0 FA
rrmov	%ecx -> %ebx	40 1 2 00
add	%eax + %ebx	00 0 1 00
push	%ebx	B1 1 0 01
irmov	FD -> %ecx	70 2 0 FD
rrmov	%ecx -> %eax	40 0 2 00
irmov	F9 -> %ecx	70 2 0 F9
rrmov	%ecx -> %ebx	40 1 2 00
add	%eax + %ebx	00 0 1 00
push	%ebx	B1 1 0 01
irmov	FC -> %ecx	70 2 0 FC
rrmov	%ecx -> %eax	40 0 2 00
irmov	F8 -> %ecx	70 2 0 F8
rrmov	%ecx -> %ebx	40 1 2 00
add	%eax + %ebx	00 0 1 00
push	%ebx	B1 1 0 01
irmov	FB -> %ecx	70 2 0 FB
rrmov	%ecx -> %eax	40 0 2 00
irmov	F7 -> %ecx	70 2 0 F7
rrmov	%ecx -> %ebx	40 1 2 00
add	%eax + %ebx	00 0 1 00
push	%ebx	B1 1 0 01



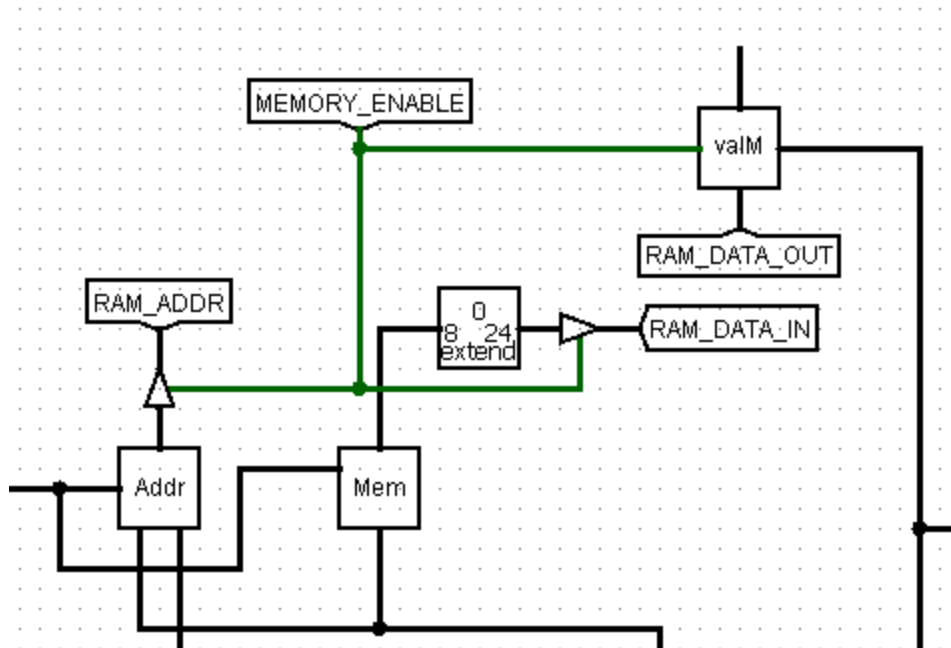
Decode: The decode step is then initiated by using the OP code to decide which sources to read from for the valA and valB busses.



Execute: Next, the execute stage begins where the values at ALU A and B are passed into the ALU. The instructed function is performed based on the iFun value. This is then stored into valE.



Memory: Next, the memory stage is initiated depending on the iCode. It either stores or loads data to/from the RAM. Fetched values are stored in valM.



Write back: Finally, the program writes back data values from valE and/or valM into the register file at the location specified by the instruction. The PC is then incremented based on whether or not a jump was encountered. The processor then loops into the same set of steps.

