Spring 2022, CS350 Final, Thur, 5/12/2022 Name:

Make sure you have 8 pages. Do not take any page(s) with you. Any missing page(s) will result in failure in the exam. This exam is closed book closed notes. Do not exchange anything during the exam. You all have the same exam. **No questions will be answered during the exam, including typos.** I don't want to give different answers to different people. If you are in doubt, briefly state your assumptions below, including typos if any. Make sure you are sitting at the designated seat. Make sure you are holding your exam, not someone else's. **You may not go to the restroom.**

Problem 1 on recursion (20 points): The code below is a recursive function unknown(x).

```
unknown:
            pushq       %rbp
            movq        %rsp, %rbp
            pushq       %rbx
            subq        $24, %rsp
            movq        %rdi, -24(%rbp)
            cmpq        $1, -24(%rbp)
            jg          .L2
            movq        -24(%rbp), %rax
            jmp         .L3
.L2:
            movq        -24(%rbp), %rax
            subq        $1, %rax
            movq        %rax, %rdi
            call        unknown
            movq        %rax, %rbx
            movq        -24(%rbp), %rax
            subq        $2, %rax
            movq        %rax, %rdi
            call        unknown
            addq        %rbx, %rax
.L3:
            addq        $24, %rsp
            popq        %rbx
            popq        %rbp
            ret
```

When x=4, list the values of parameter passed each time unknown is called. List them in the order in which unknown is called.

When x=3, find the number of iterations including the very first one.

When x=3, find the return value.

Problem 2 on sequential computer (20 points): The instructions shown below with various values below are executed on the sequential computer discussed in class. **Fill the 20 table entries, a-t, with values**. ***All instructions are independent***. Write N/A or leave blank if not applicable or not available.

Memory addresses each are 8 bytes (64 bits).	rax=1, rbx=2, rdx=0x1000, rsp=0x2000
All instructions are at memory address 0x0000.	loop is at 0x3000
memory 0x1000: 1 (64 bits)	func is at 0x4000
memory 0x2000: 2 (64 bits)	previous instructions resulted in greater than zero.
memory 0x3000: 3 (64 bits)	
memory 0x4000: 4 (64 bits)	

Instruction	icode	ifun	rA	rB	valC	valA	valB	dstE	dstM	srcB	valE	valM	Cnd
addq %rax,%rbx						**e**	**f**	**g**	**h**				
mrmovq 5(%rbx),%rax	**a**	**b**	**c**	**d**									
push %rax						**m**	**n**	**o**	**p**				
jmp loop	**i**	**j**	**k**	**l**									
call func					**q**	**r**	**s**	**t**					

Problem 3 on pipelined computer with branch misprediction (20 points): Fill the 20 table entries, a-t, with values when the instructions below are executed on the pipeline discussed in class. ***Instructions depend on the previous one***. Write N/A or leave blank if not applicable or not available.

```
0x000:   xorq %rax,%rax                       0x017:   nop
0x002:   jne  t             # Not taken       0x018:   halt
0x00b:   irmovq $1, %rax    # Fall through    0x019: t: irmovq $3, %rdx   # Target (Should not execute)
0x015:   nop                                  0x023:   irmovq $4, %rcx   # Should not execute
0x016:   nop                                  0x02d:   irmovq $5, %rdx   # Should not execute
```

clock	1		2		3		4		5	
	fetch		**decode**		**execute**		**memory**		**writeback**	
	predPC	**a**	icode		icode		icode	**c**	icode	
			ifun		ifun		Cnd	**d**	valE	
			rA		rA		valE	**e**	valM	
			rB		rB		valA		dstE	
xor rax,rax			valC		valC		dstE	**f**	dstM	
			valP	**b**	valA		dstM			
					valB					
					dstE					
					dstM					
					srcA	**g**				
					srcB	**h**				
			fetch		**decode**		**execute**		**memory**	
			predPC	**i**	icode	**j**	icode		icode	
					ifun	**k**	ifun		Cnd	**n**
					rA		rA		valE	
					rB		rB		valA	**o**
jne t					valC	**l**	valC		dstE	
					valP	**m**	valA		dstM	
							valB			
							dstE			
							dstM			
							srcA			
							srcB			
					fetch		**decode**		**execute**	
					predPC	**p**	icode		icode	**q**
							ifun		ifun	**r**
You need to figure out the instruction here							rA		rA	**s**
							rB		rB	**t**
							valC		valC	
							valP		valA	
									valB	
									dstE	
									dstM	
									srcA	
									srcB	

Problem 4 on cache memory (10 points): The following problem concerns basic cache lookups. The memory is byte addressable. Memory accesses are to 1-byte words (not 4-byte words). Physical addresses are 12 bits wide. The cache is 4-way set associative, with a 2-byte block size and 32 total lines. In the following tables, all numbers are given in hexadecimal. The contents of the cache are as follows:

Index	Tag	Valid	Byte0	Byte1	Tag	Valid	Byte0	Byte1	Tag	Valid	Byte0	Byte1	Tag	Valid	Byte0	Byte1
0	29	0	34	29	87	0	39	AE	7D	1	68	F2	8B	1	64	38
1	F3	1	0D	8F	3D	1	0C	3A	4A	1	A4	DB	D9	1	A5	3C
2	A7	1	E2	04	AB	1	D2	04	E3	0	3C	A4	01	0	EE	05
3	3B	0	AC	1F	E0	0	B5	70	3B	1	66	95	37	1	49	F3
4	80	1	60	35	2B	0	19	57	49	1	8D	0E	00	0	70	AB
5	EA	1	B4	17	CC	1	67	DB	8A	0	DE	AA	18	1	2C	D3
6	1C	0	3F	A4	01	0	3A	C1	F0	0	20	13	7F	1	DF	05
7	0F	0	00	FF	AF	1	B1	5F	99	0	AC	96	3A	1	22	79

A: The bits for block offset are
(a) bit0-bit3 (b)bit4-bit5 (c)bit6-bit8 (d)bit9-bit11 (e) None of the above

B: The bits for index are
(a) bit0-bit3 (b)bit1-bit3 (c)bit4-bit6 (d)bit4-bit7 (e) None of the above

C: The bits for tag are
(a) bit0-bit6 (b)bit3-bit10 (c)bit4-bit11 (d)bit5-bit8 (e) None of the above

D: The cache offset value for the physical address 0xEAA in hexa is
(a) 0 (b)1 (c)2 (d)3 (e) None of the above

E: The cache index value for the physical address 0xEAA in hexa is
(a) 3 (b)4 (c)5 (d)6 (e) None of the above

F: The cache tag value for the physical address 0xEAA in hexa is
(a) EA (b)AA (c) BA (d)EB (e) None of the above

G: The physical address 0xEAA results in
(a)miss (b)B4 (c)17 (d)67 (e) None of the above

H: The cache index value for the physical address 0xABC in hexa is
(a)A (b) B(c)C (d)D (e) None of the above

I: The cache tag value for the physical address 0xABC in hexa is
(a)AB (b)AC (c)BC (d)AC (e) None of the above

J: The physical address 0xABC results in
(a)miss (b)04 (c)66 (d)AB (e) None of the above

Problem 5 on cache performance (10 points): Consider a function that initializes a 960x1280 array of structs each of which has eight chars to 0. Assume the machine has a 128 KB direct mapped cache with 8 byte lines and the C structures you are using are:

```
struct mystrct {    char a, b, c, d, e, f, g, h; };
struct mystrct myarray[960][1280];
register int i, j;
register char *cptr;
register int *iptr;
```

Assume sizeof(char) = 1, sizeof(int) = 4, myarray begins at memory address 0, the cache is initially empty, the only memory accesses are to the entries of the array myarray, and variables i, j, cptr, and iptr are stored in registers.

A: The cache miss rate (%) for the code below is

```
for (j=0; j < 1280; j++)
   for (i=0; i < 960; i++){ myarray[i][j].a = 0; myarray[i][j].c = 0; myarray[i][j].e = 0; myarray[i][j].f = 0; }
```

(a) 0 (b)12.5 (c)25 (d)50 (e) 75 (f) 87.5 (g) None of the above

B: The cache miss rate (%) for the code below is

```
for (j=0; j < 1280; j++) {
   for (i=0; i < 960; i++){
      myarray[i][j].a = 0; myarray[i][j].b = 0; myarray[i][j].c = 0; myarray[i][j].d = 0;
      myarray[i][j].e = 0; myarray[i][j].f = 0; myarray[i][j].g = 0; myarray[i][j].h = 0;
   }
}
```

(a) 0 (b)12.5 (c)25 (d)50 (e) 75 (f) 87.5 (g) None of the above

C: The cache miss rate (%) for the code below is

```
char *cptr;
cptr = (char *) myarray;
for (; cptr < (((char *) myarray) + 1280 * 960 * 8); cptr++) *cptr = 0;
```

(a) 0 (b)12.5 (c)25 (d)50 (e) 75 (f) 87.5 (g) None of the above

D: The cache miss rate (%) for the code below is

```
int *iptr;
iptr = (int *) myarray;
for (; iptr < (myarray + 1280 * 960); iptr++) *iptr = 0;
```

(a) 0 (b)12.5 (c)25 (d)50 (e) 75 (f) 87.5 (g) None of the above

E: Which code among the above four A, B, C, and D, should be the slowest?

(a) A (b)B (c)C (d)D (e) C and D (f)Cannot determine

Problem 6 on locality of reference - the book cover (20 points): The textbook cover is shown below, where A-D represents ridges while w-z valleys. **Each wrong answer will get -2 points.** No answers will get zero point.

1) A to D represents temporal locality: True False

2) w to z represents spatial locality: True False

3) C represents L2 cache: True False

4) A represents memory: True False

5) Temporal locality exists along the size axis: True False

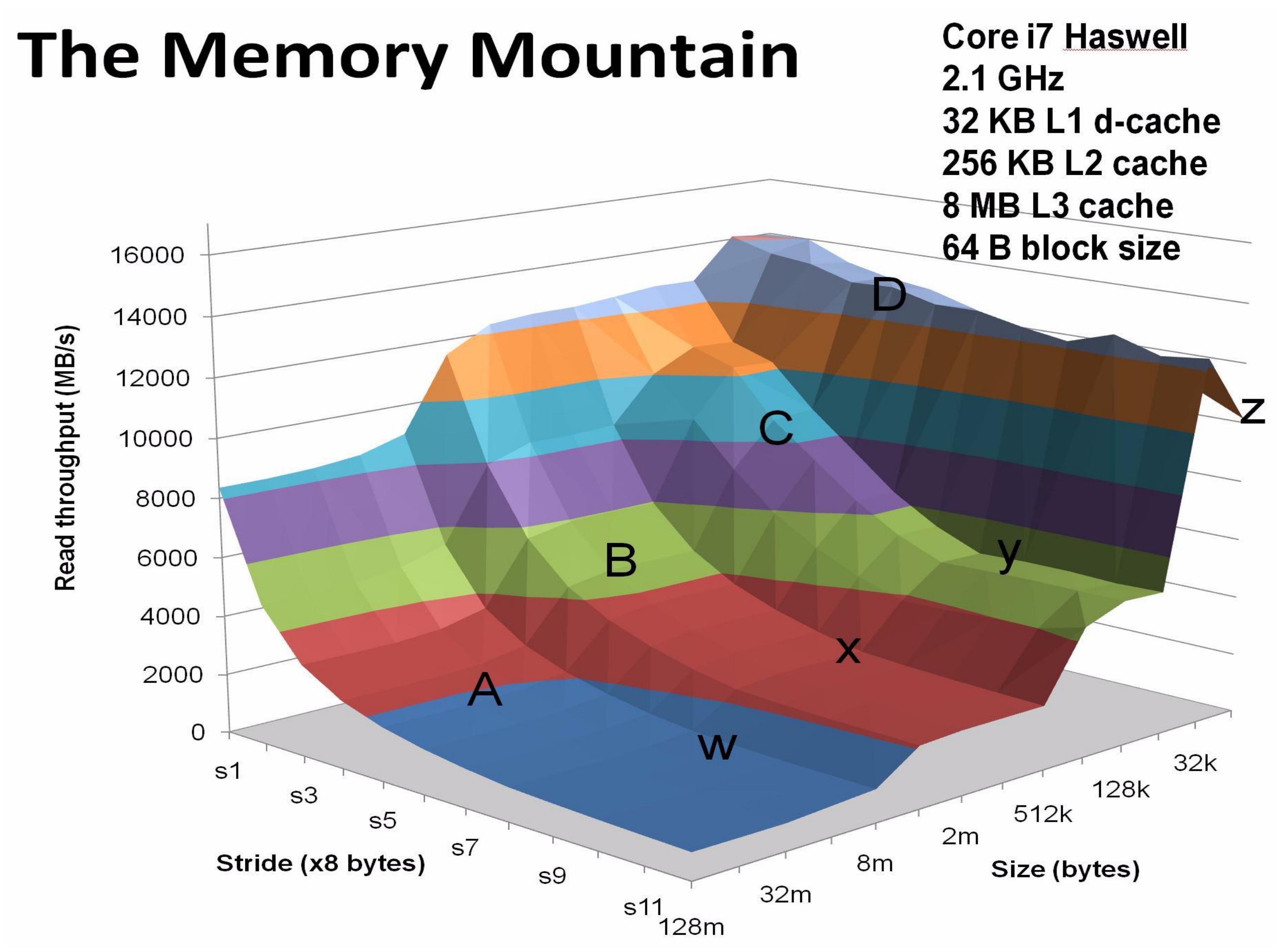
6) Spatial locality exists along the size axis: True False

7) D demonstrates stride can be made relevant: True False

8) C demonstrates stride can be made relevant: True False

9) A demonstrates that stride can be made irrelevant: True False

10) A to w, B to x, C to y and D to z (ridge to valley) demonstrate that size can be made irrelevant: True False



Y86-64 Instruction Set

Instruction	Byte 0		Byte 1		2	3	4	4	5	6	7	8	B9
	icode	fn	rA	rB									
halt	0	0											
nop	1	0											
cmove rA, rB	2	3	rA	rB									
rrmovq rA, rB	2	0	rA	rB									
irmovq V, rB	3	0	F	rB					V				
rmmovq rA, D(rB)	4	0	rA	rB					D				
mrmovq D(rB), rA	5	0	rA	rB					D				
addq rA, rB	6	0	rA	rB									
sub		1											
and		2											
xor		3											
jmp Dest	7	0					Dest						
jle		1											
jl		2											
je		3											
jne		4											
jge		5											
jg		6											
call Dest	8	0					Dest						
ret	9	0											
pushq rA	A	0	rA	F									
popq rA	B	0	rA	F									

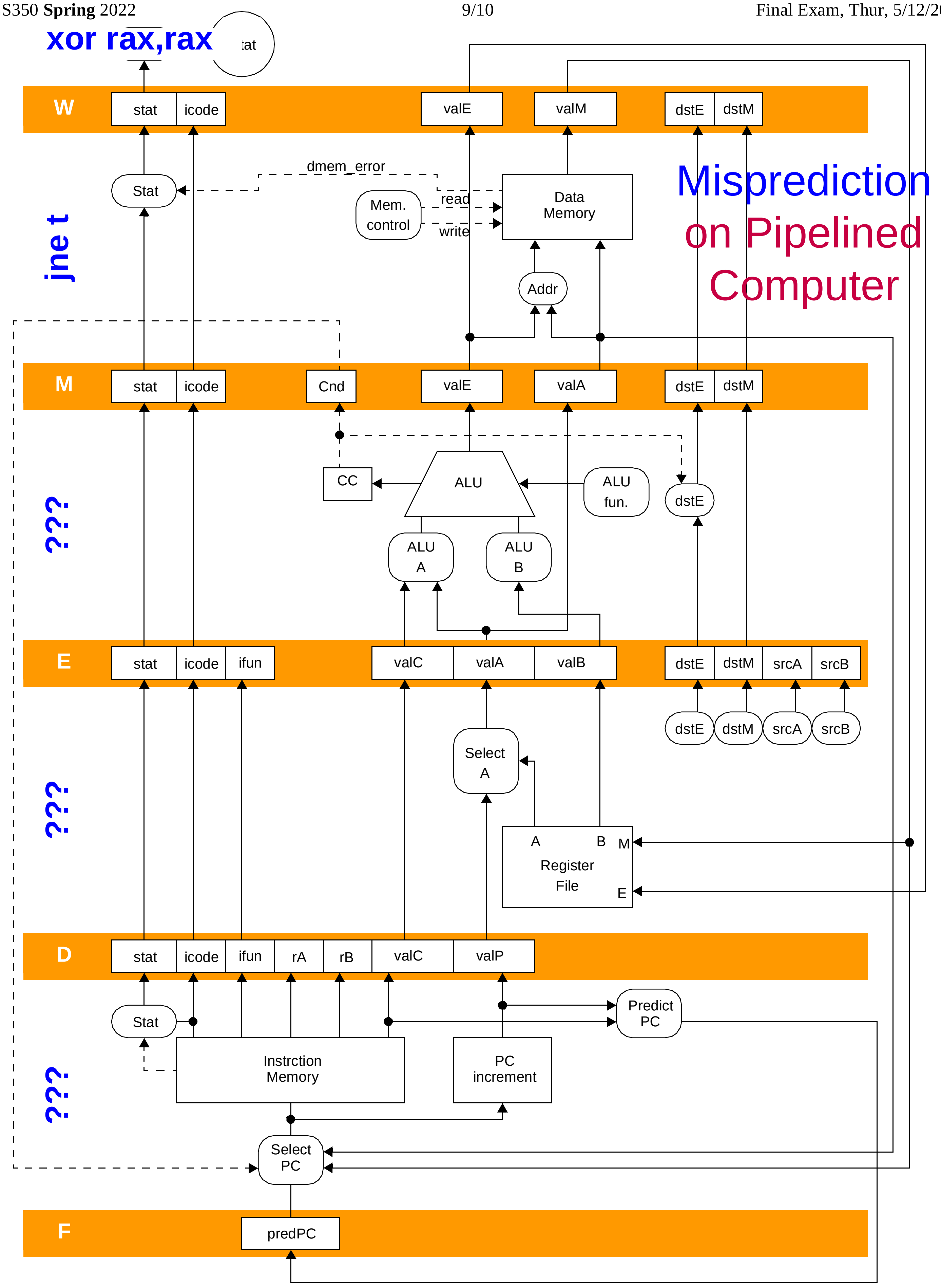
Register assignment

rax	rcx	rdx	rbx	rsp	rbp	rsi	rdi	r8	r9	r10	r11	r12	r13	r14	no register
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

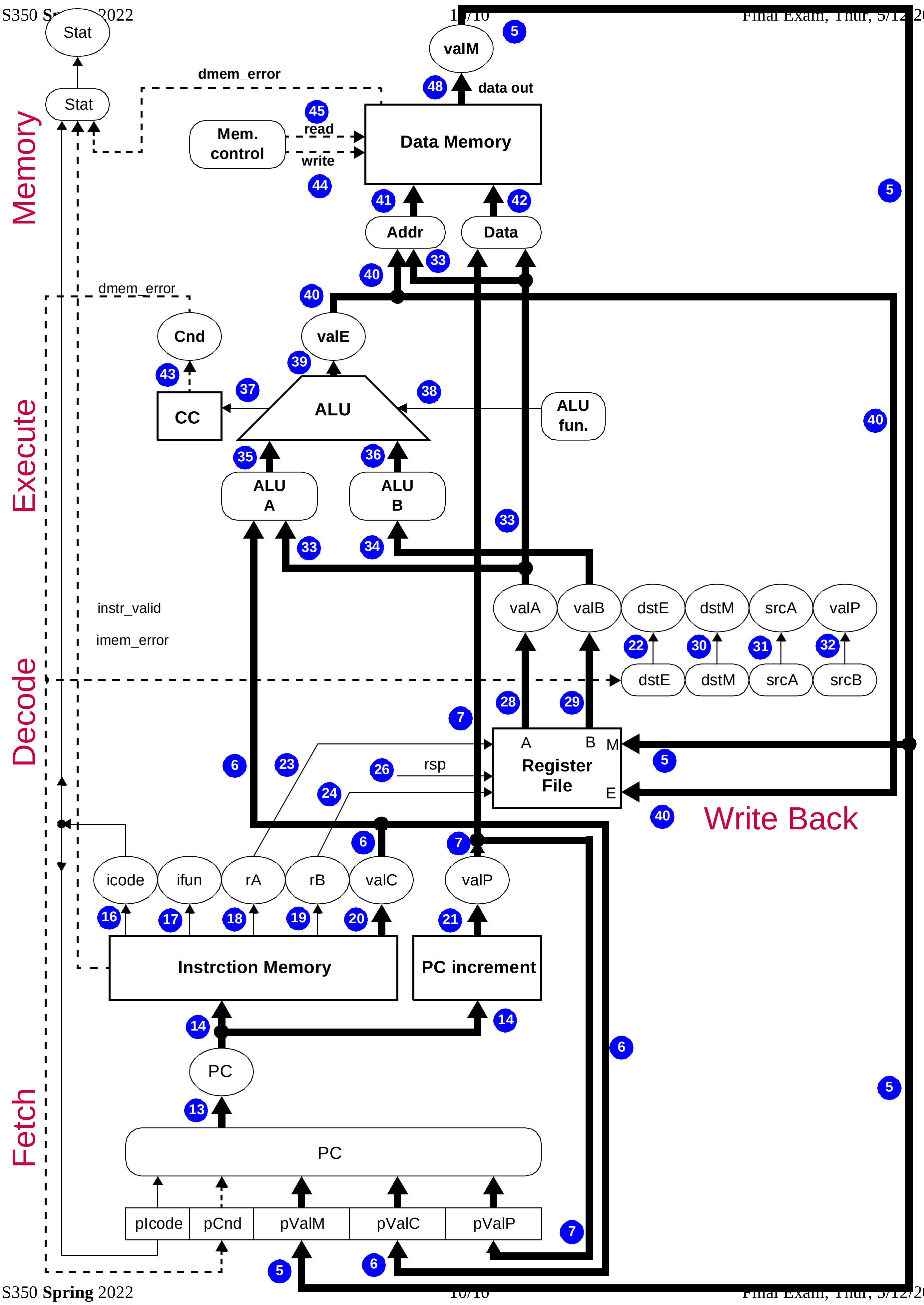
Definition of Instructions

		ALU Op	**reg to reg**	**imm to reg**	**load**	**store**
		OPq rA, rB	rrmovq rA, rB	irmovq V, rB	mrmovq D(rB), rA	rmmovq rA, D(rB)
F	icode,ifun	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]
	rA,rB	rA:rB <- M1[PC+1]	rA:rB <- M1[PC+1]	rA:rB <- M1[PC+1]	rA:rB <- M1[PC+1]	rA:rB <- M1[PC+1]
	valC			valC <- M8[PC+2]	valC <- M8[PC+2]	valC <- M8[PC+2]
	valP	valP <- PC+2	valP <- PC+2	valP <- PC+10	valP <- PC+10	valP <- PC+10
D	valA, srcA	valA <- R[rA]	valA <- R[rA]			valA <- R[rA]
	valB, srcB	valB <- R[rB]			valB <- R[rB]	valB <- R[rB]
EX	valE	valE <- valB OP valA	valE <- 0 + valA	valE <- 0 + valC	valE <- valB + valC	valE <- valB + valC
	Cond code	Set CC				
M	valM				valM<- M8[valE]	M8[valE] <- valA
WB	dstE	R[rB] <- valE	R[rB] <- valE	R[rB] <- valE	R[rA] <- valM	
	dstM					
PC	PC	PC <- valP	PC <- valP	PC <- valP	PC <- valP	PC <- valP

		push	**pop**	**jmp**	**call**	**ret**	**cmov**
		pushq rA	popq rA	jXX Dest	call Dest	ret	cmovXX rA, rB
F	icode,ifun	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]	icode:ifun <- M1[PC]
	rA,rB	rA:rB <- M1[PC+1]	rA:rB <- M1[PC+1]				rA:rB <- M1[PC+1]
	valC			valC <- M8[PC+1]	valC <- M8[PC+1]		
	valP	valP <- PC+2	valP <- PC+2	valP <- PC+9	valP <- PC+9	valP <- PC+1	valP <- PC+2
D	valA, srcA	valA <- R[rA]	valA <- R[%rsp]			valA <- R[%rsp]	valA <- R[rA]
	valB, srcB	valB <- R[%rsp]	valB <- R[%rsp]		valB <- R[%rsp]	valB <- R[%rsp]	valB <- 0
EX	valE	valE <- valB + – 8	valE <- valB + 8		valE <- valB + –8	valE <- valB + 8	valE <- valB + valA
	Cond code			Cnd <- Cond(CC,ifun)			If ! Cond(CC,ifun) rB <- 0xF
M	valM	M8[valE] <- valA	valM<- M8[valA]		M8[valE] <- valP	valM<- M8[valA]	
WB	dstE	R[%rsp] <- valE	R[%rsp] <- valE		R[%rsp] <- valE	R[%rsp] <- valE	R[rB] <- valE
	dstM		R[rA] <- valM				
PC	PC	PC <- valP	PC <- valP	PC <- Cnd ? valC : valP	PC <- valC	PC <- valM	PC <- valP



xor rax,rax
tat
W
stat
icode
valE
valM
dstE
dstM
dmem_error
Stat
Mem. control
read
write
Data Memory
Addr
Misprediction
on Pipelined
Computer
jne t
M
stat
icode
Cnd
valE
valA
dstE
dstM
???
CC
ALU
ALU fun.
dstE
ALU A
ALU B
E
stat
icode
ifun
valC
valA
valB
dstE
dstM
srcA
srcB
dstE
dstM
srcA
srcB
Select A
???
A
B
M
Register File
E
D
stat
icode
ifun
rA
rB
valC
valP
Predict PC
Stat
???
Instrction Memory
PC increment
Select PC
F
predPC


Memory
Execute
Decode
Fetch
Write Back
Stat
Stat
dmem_error
valM
data out
Mem. control
read
write
Data Memory
Addr
Data
dmem_error
Cnd
valE
CC
ALU
ALU fun.
ALU A
ALU B
instr_valid
imem_error
valA
valB
dstE
dstM
srcA
valP
dstE
dstM
srcA
srcB
A
B
M
E
Register File
rsp
icode
ifun
rA
rB
valC
valP
Instrction Memory
PC increment
PC
PC
pIcode
pCnd
pValM
pValC
pValP
5
48
45
44
41
42
33
40
40
39
43
37
38
35
36
33
34
33
40
22
30
31
32
28
29
7
6
23
26
24
5
40
6
7
16
17
18
19
20
21
14
14
6
13
5
7
5
6
5