4190.308: Computer Architecture
Final Exam

December 16th, 2016

Professor Jae W. Lee

SOLUTIONS

Student ID #: ____________________

Name: ____________________

This is a closed book, closed notes exam.

120 Minutes

16 Pages

(+ 2 Appendix Pages)

Total Score: 200 points

Notes:

- Please turn off all of your electronic devices (phones, tablets, notebooks, netbooks, and so on). A clock is available on the lecture screen.
- Please stay in the classroom until the end of the examination.

- You must not discuss the exam's contents with other students during the exam.
- You must not use any notes on papers, electronic devices, desks, or part of your body.

(This page is intentionally left blank. Feel free to use as you like.)

Part A: Short Answers (20 points)

Question 1 (20 points)

Please answer the following questions. You don’t have to justify your answer—just write down your answer only.

Do not guess. You will get 4 points for each correct answer and lose 4 points for each wrong answer (but 0 point for no answer).

(1) CISC architectures (e.g., Intel x86-64) generally have an advantage in code size over RISC architectures (e.g., MIPS, ARM). (True/False) TRUE

(2) If two processors implement the same ISA, one with higher clock frequency *always* performs better. (True/False) FALSE

(3) Instruction pipelining not only increases throughput but also reduces the latency of each individual instruction. (True/False) FALSE

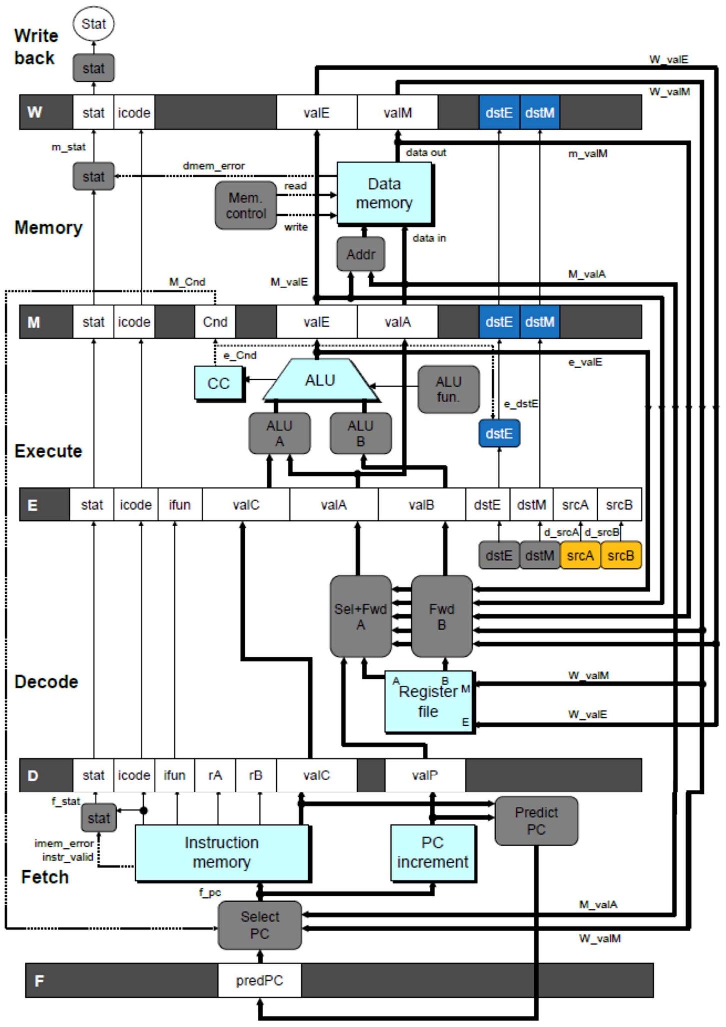
(4) Assuming the cache size is fixed, the number of index bits decreases as we increase the associativity. (True/False) TRUE

(5) Write down three types of cache misses. Compulsory, Conflict, Capacity

Part B: Pipelining (32 points)

Question 2 (32 points)

SNU Electronics Inc. (SEI) has two product lines of Y86-64 compatible processors: low-end *s3*^{TM} processor family and high-end *s7*^{TM} processor family. The figure below shows the pipeline of the *s7*^{TM} processor family, which implements full forwarding logic.



SEI's high-end *s7*^{TM} processor pipeline

We will evaluate the performance of both processors using the following test code:

```
I1: popq     %rax
I2: addq     %r10, %rax
I3: xorq     %rax, %rax
I4: je       I6                # branch to I6 if equal-to-zero
I5: mrmovq   (%rbx), %r11
I6: irmovq   $2016, %rdx
I7: subq     %rdx, %rax
```

(1) The *s3*^{TM} processor is a stripped-down version of *s7*^{TM}, targeting the low-cost market with the following features:

- Standard five (5) stage (F, D, E, M, W) pipeline
- **No forwarding logic**
- Stalls on all data and control hazards
- **Instructions are not fetched until branch condition is checked.**
- The same register CAN be read & written on the **same** clock cycle.

Count how many cycles will be needed to execute the test code on *s3*^{TM} by writing out each instruction's progress through the pipeline by filling in the table below with pipeline stages (F: Fetch, D: Decode, E: Execute, M: Memory, W: Write Back).

Notes: Don't show any instruction that was not executed.

Cycle / Instr	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1	**F**	**D**	**E**	**M**	**W**															
I2		**F**	**D**	**D**	**D**	**E**	**M**	**W**												
I3			**F**	**F**	**F**	**D**	**D**	**D**	**E**	**M**	**W**									
I4						**F**	**F**	**F**	**D**	**E**	**M**	**W**								
I6									**-**	**-**	**F**	**D**	**E**	**M**	**W**					
I7												**F**	**D**	**D**	**D**	**E**	**M**	**W**		

	F	D	E	M	W
1	I1				
2	I2	I1			
3	I3	I2	I1		
4	I3	I2	B	I1	
5	I3	I2	B	B	I1
6	I4	I3	I2	B	B
7	I4	I3	B	I2	B
8	I4	I3	B	B	I2
9	I5	I4	I3	B	B
10	I5	B	I4	I3	B
11	I6(~~I5~~)	B	B	I4	I3
12	I7	I6	B	B	I4
13		I7	I6	B	B
14		I7	B	I6	B
15		I7	B	B	I6
16			I7	B	B
17				I7	B
18					I7

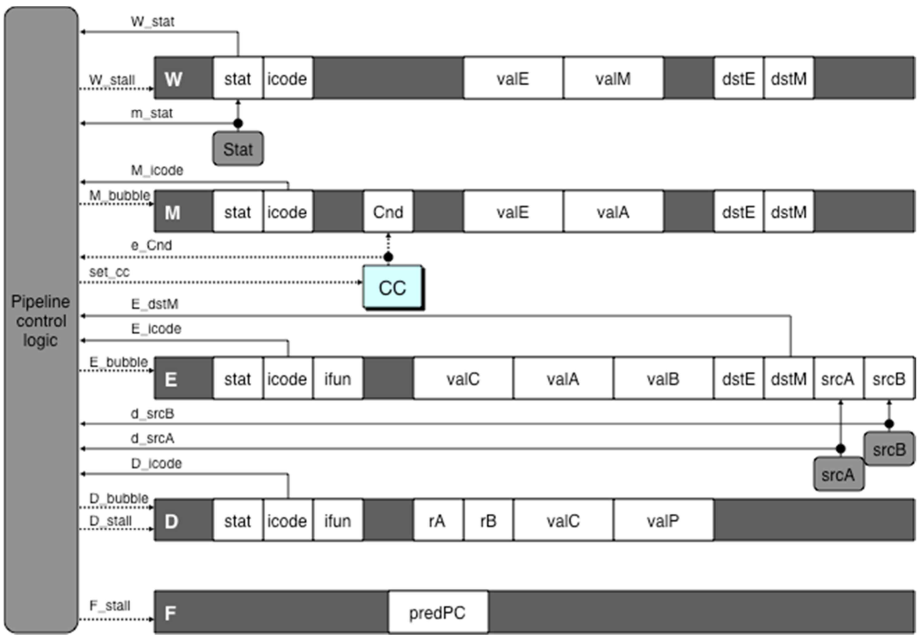
(2) The high-end *s7*^{TM} processor targets high-performance market with the following features:

- Standard five (5) stage (F, D, E, M, W) pipeline
- **Forwarding logic for data and control hazard**
- **Branch prediction strategy**
    - **Conditional jumps: *never taken* (NT)**
    - **Call and unconditional jumps: to predict next PC to be destination**
    - **No prediction for return instruction**
- The same register CAN be read & written on the **same** clock cycle.

Cycle / Instr	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
I1	**F**	**D**	**E**	**M**	**W**															
I2		**F**	**D**	**D**	**E**	**M**	**W**													
I3			**F**	**F**	**D**	**E**	**M**	**W**												
I4					**F**	**D**	**E**	**M**	**W**											
I6						**-**	**-**	**F**	**D**	**E**	**M**	**W**								
I7									**F**	**D**	**E**	**M**	**W**							

	F	D	E	M	W
1	I1				
2	I2	I1			
3	I3	I2	I1		
4	I3	I2	B	I1	
5	I4	I3	I2	B	I1
6	I5	I4	I3	I2	B
7	I6	I5	I4	I3	I2
8	I6	B	B	I4	I3
9	I7	I6	B	B	I4
10		I7	I6	B	B
11			I7	I6	B
12				I7	I6
13					I7

(3) Here is a figure of the pipeline control logic of *s7*^{TM}. Fill out the HCL code of the four control signals below. (`e_Cnd`: Boolean value showing whether a branch is taken or not.)



```
bool F_stall =
    E_icode in {IMRMOVQ, IPOPQ} &&
    (E_dstM == d_srcB || E_dstM == d_srcA) ||
    IRET in {D_icode, E_icode, M_icode};
```

```
bool D_stall =
    E_icode in {IMRMOVQ, IPOPQ} &&
    (E_dstM == d_srcB || E_dstM == d_srcA);
```

```
bool D_bubble =
    (E_icode == IJXX && E_ifun != 0 && e_Cnd) ||
    !(E_icode in {IMRMOVQ, IPOPQ} &&
      E_dstM in {d_srcA, d_srcB}) &&
    IRET in {D_icode, E_icode, M_icode};
```

```
bool E_bubble =
    (E_icode == IJXX && E_ifun != 0 && e_Cnd) ||
    E_icode in {IMRMOVQ, IPOPQ} &&
    (E_dstM == d_srcB || E_dstM == d_srcA);
```

Part C: Caches (38 points)

Question 3 (14 points)

Let's assume that the new 32-bit address layout taking the index from the upper bits instead of lower bits.

31 22	21 6	5 0
Index	Tag	Offset
10 bits	16 bits	6 bits

For the following C program with a direct mapped cache, the cache hit rate will increase/decrease/not change (**select one**) by adopting new address layout. Explain why in one paragraph.

```
int  A[1024];   // 4 KB array

for (i=0; i<M; i++)
  for (j=0; j<1024; j++)
    read(A[j]);  // accessing array element A[j]
```

(4 points) Answer
Cache hit rate will decrease (i.e., cache will perform worse).

(10 points) Reason
It's because all cache lines in A[] will be mapped to the same line to cause excessive conflict misses.

Question 4 (24 points)

In this question we want to calculate the cache miss rate for a given program analytically. Consider the following cache configuration (data cache only):

Parameter	Value
Cache size	32 KB (2^{15} bytes)
Cache block size (B)	64 bytes
Associativity (E)	1 (direct-mapped)
Size of `int` data	4 bytes

For each of the following programs, calculate the *average* cache miss rate. Assume there are no other memory accesses than those to array `A[]`. (8pts each)

(a) Assume the cache is initially empty.

```
int  A[16384];   // 64 KB array

for (i=0; i<16384; i++)
  read(A[i]);    // accessing array element A[i]
```

Cache miss rate: 1/16

(b) Assume the cache is initially empty.

```
int  A[16384];   // 64 KB array

for (i=0; i<16384; i++) {
  int my_idx = (i*16) % 16384;

  // accessing array element A[my_idx]
  read(A[my_idx]);
}
```

Cache miss rate: 1

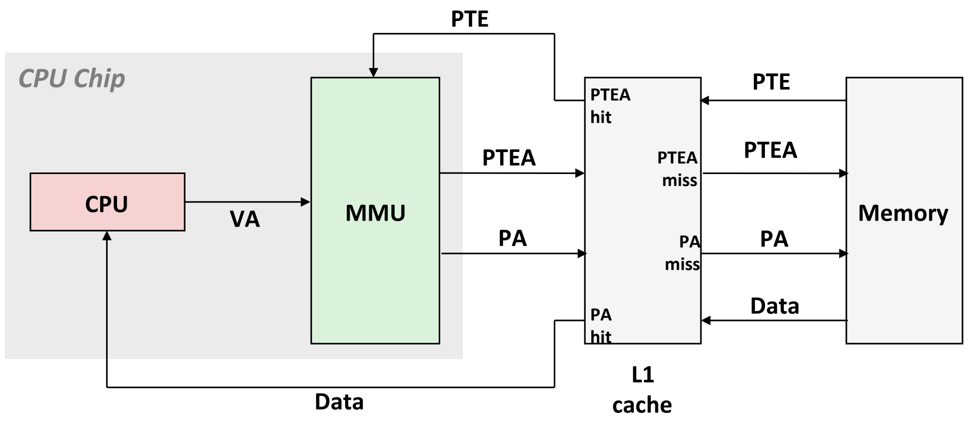
(c) Assume the cache is initially filled with the first half of array `A[]`.

```
int  A[16384];  // 64 KB array

for (i=0; i<16384; i++) {

  // random index between 0 and 16384
  int my_idx = random() % 16384;

  // accessing array element A[my_idx]
  read(A[my_idx]);
}
```

Cache miss rate: 1/2

Part D: Cache and Virtual Memory (92 points)

The figure below shows the access path of a physically addressed cache integrated with VM.



Question 5 (16 points)

To speed up L1 cache accesses, modern CPUs implement *virtually indexed physically tagged cache*. How does a virtually indexed physically tagged cache work and why it has shorter access time than the physically addressed cache shown above? Write your answer in one paragraph.

When (cache index) is contained in VPO (virtual page offset), we can get a CI from VA (virtual address) before address translation. Because a virtually indexed physically tagged cache can parallelize the address translation and the CT (cache tag) lookup using the CI from VA, it access has shorter access time.

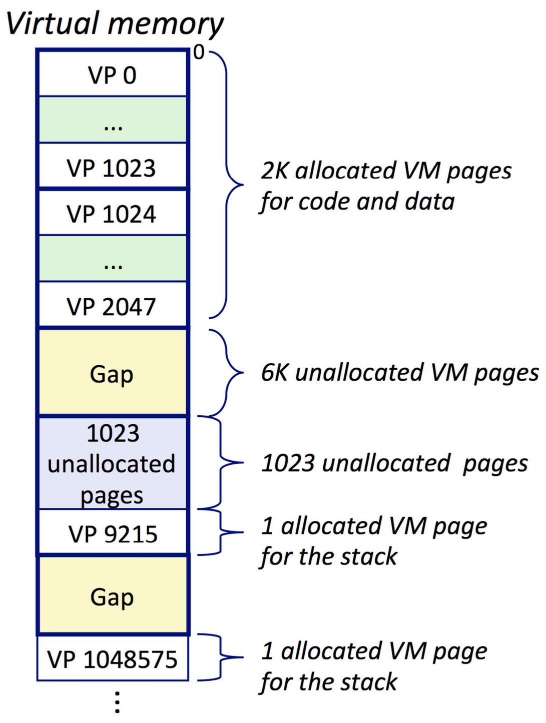
Question 6 (16 points)

In a physically addressed cache, a memory reference can encounter three different types of misses: a TLB miss, a page fault, and a cache miss. Consider all the combination of these three events with one or more occurring (Case 1-7). Identify **all** possible cases that can occur among the seven cases. (*Note*: You don't have to justify your answer.)

Case #	TLB	Page table	Cache
1	**Hit**	**Hit**	**Miss**
2	Hit	Miss	Hit
3	Hit	Miss	Miss
4	**Miss**	**Hit**	**Hit**
5	**Miss**	**Hit**	**Miss**
6	Miss	Miss	Hit
7	**Miss**	**Miss**	**Miss**

Question 7 (24 points)

Assuming that a process is using the virtual pages 0 ~ 2047, 9215, and 1048575, as shown in the figure below. The address space is 32-bit, a page is 4KB, and a page table entry takes 4 bytes. Please answer the following questions.



1) What is the size of the page table if a one-level linear page table is used?

   (12pts) $2^{32} * 2^{-12} * 2^{2} = 2^{22} = 4194304 = 4$ MB

2) What is the size of the page table if a 2-level page table is used? Assume the L1 page table has 1024 ($=2^{10}$) entries.

   (12pts)
   1-level: 1024 * 4B = 4 KB
   2-level: 4 * 4KB = 16 KB
   TOTAL: 20 KB

Question 8 (36 points)

For the rest of this question, please assume the following

- Page size = 64 bytes
- 14-bit virtual address
- 12-bit physical address
- 4-way set associative TLB with 16 entries
- 2-way set associative, physically addressed cache with 16 bytes/block

We ask you to follow step-by-step operations of a 2-way physically addressed cache.

Set	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid	Tag	PPN	Valid
0	-	-	0	03	0D	1	01	18	1	04	34	1
1	09	3E	1	-	-	0	02	04	1	0D	02	1
2	02	03	1	-	-	0	01	22	1	-	-	0
3	02	3B	1	03	2F	1	01	16	1	-	-	0

Initial TLB state (16 entries)

Index	Tag	Valid	Tag	Valid
0	14	1	-	0
1	0C	1	-	0
2	16	1	-	0
3	0F	1	-	0
4	14	1	-	0
5	01	1	-	0
6	1F	1	-	0
7	17	1	-	0

Initial cache state (16 entries, only tag and valid bits are shown)

(1) Assume we access the cache with virtual address `0x3F0`. Please fill out the blank in **hexadecimal number**.
(*Note*: You should write '`X`' if the value would be not specified.)

VPN:	0xF	PPN:	0x2F
TLBI:	0x3	CO (cache offset):	0x0
TLBT:	0x3	CI (cache index):	0x7
TLB hit? (Y/N)	Y	CT (cache tag):	0x17
		Cache hit? (Y/N)	Y

(2) Assuming the Least Recently Used (LRU) replacement policy, what will be the final cache state after accessing the following address sequence? Fill out the table below with the new cache state.

Address sequence (all in *virtual* address)**:**
`0x974` → `0xD7C` → `0x110` → `0x310` → `0x29C` → `0x2E8` → `0xD70` → `0x1C4`

Index	Tag	Valid	Tag	Valid
0	14	1	**0B**	**1**
1	0C	1	-	0
2	16	1	-	0
3	**01**	**1**	**1F**	**1**
4	14	1	-	0
5	01	1	**06**	**1**
6	1F	1	**1D**	**1**
7	17	1	-	0

(3) What will be the cache hit rate for (2)?
M → M → H → M → H → M → H → M, 3/8 = **37.5%**

VA	VA (bin)				PA (bin)		
(hex)	TLBT	TLBI	PO		CT	CI	CO
`974`	`1001`	`01`	`110100`		`11111`	`011`	`0100`
`D7C`	`1101`	`01`	`111100`		`00001`	`011`	`1100`
`110`	`0001`	`00`	`010000`		`01100`	`001`	`0000`
`310`	`0011`	`00`	`010000`	→	`00110`	`101`	`0000`
`29C`	`0010`	`10`	`011100`		`00001`	`101`	`1100`
`2E8`	`0010`	`11`	`101000`		`11101`	`110`	`1000`
`D70`	`1101`	`01`	`110000`		`00001`	`011`	`0000`
`1C4`	`0001`	`11`	`000100`		`01011`	`000`	`0100`

Part E: Performance (18 points)

Question 9 (18 points)

You are a manager responsible for deciding which processor to use for your company's new gaming console. The gaming console is required to run a killer application very fast (titled *Hungry Bird*^{TM}). The two candidate processors (P_A and P_B) have different ISAs. The table below summarizes the performance analysis results of the application on the two processors.

Instruction Type	**Instr. count (millions)**		**Cycles per Instr. (CPI)**	
	P_A	**P_B**	**P_A**	**P_B**
Arithmetic & Logic	15	20	1	1
Load & Store	3	6	5	4
Branch	2	4	10	7

Clock rate	
P_A	**P_B**
800 MHz	1 GHz

(1) What are the average CPIs for this app on both processors? (8pts)

P_A = 2.5
P_B = 2.4

(2) What are the CPU times of this app on both processors? (10pts)
(Note: Be sure to include time units.)

P_A = 62.5 ms
P_B = 72 ms

(This page is intentionally left blank. Feel free to use as you like.)

Appendix A: Y86-64 (Instruction Set)

Instruction	icode:fn		rA:rB		
byte	0		1		2 3 4 5 6 7 8 9
halt	0 = IHALT	0			
nop	1 = INOP	0			
cmovXX rA, rB	2 = IRRMOVQ	fn			
rrmovq		0			
cmovle		1			
cmovl		2			
cmove		3			
cmovne		4			
cmovge		5			
cmovg		6			
					9
irmovq V, rB	3 = IIRMOVQ	0	F	rB	V
rmmovq rA, D(rB)	4 = IRMMOVQ	0	rA	rB	D
mrmovq D(rB), rA	5 = IMRMOVQ	0	rA	rB	D
OPq rA, rB	6 = IOPQ	fn	rA	rB	
addq		0			
subq		1			
andq		2			
xorq		3			
					8
jXX Dest	7 = IJXX	fn	Dest		
jmp		0			
jle		1			
jl		2			
je		3			
jne		4			
jge		5			
jg		6			
					8
call Dest	8 = ICALL	0	Dest		
ret	9 = IRET	0			
pushq rA	A = IPUSHQ	0	rA	F	
popq rA	B = IPOPQ	0	rA	F	

Register encoding

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

Appendix B: x86-64 (Instruction Set)

Common instructions

```
mov    src, dst    dst = src
movsbl src, dst    byte to int, sign-extend
movzbl src, dst    byte to int, zero-fill
lea   addr, dst    dst = addr

add    src, dst    dst += src
sub    src, dst    dst -= src
imul   src, dst    dst *= src
neg    src, dst    dst = -dst(arith inverse)

sal  count, dst    dst <<= count
sar  count, dst    dst >>= count(arith shift)
shr  count, dst    dst >>= count(logical shift)
and    src, dst    dst &= src
or     src, dst    dst |= src
xor    src, dst    dst ^= src
not    dst         dst = ~dst(bitwise inverse)

cmp    a, b        b - a, set flag
test   a, b        a & b, set flag

jmp    label       jump to label(unconditional)
je     label     ZF  equal/zero
jne    label    ~ZF  not equal/zero
js     label     SF        negative
jns    label    ~SF           nonnegative
jg     label   ~(SF^OF)&~ZF   greater(signed)
jge    label   ~(SF^OF)   greater or
equal(signed)
jl     label    (SF^OF)   less(signed)
jle    label    (SF^OF)|ZF   less or
equal(signed)
ja     label    ~CF&~ZF   above(unsigned)
jb     label     CF  below(unsigned)

push   src         add to top of stack
                   Mem[--%rsp] = src
pop    dst         remove top from stack
                   dst = Mem[%rsp++]
call   fn          push %rip, jump to fn
ret                pop %rip
```

Instruction suffixes

b	byte
w	word; 2 bytes
l	double word; 4 bytes
q	quad word; 8 bytes

Suffix is elided when can be inferred from operands. e.g. %rax implies q, %eax implies l

Condition codes / flags

ZF	Zero flag
SF	Sign flag
CF	Carry flag
OF	Overflow flag

Registers

%rip	Instruction pointer
%rsp	Stack pointer
%rax	Return value
%rdi	1st argument
%rsi	2nd argument
%rdx	3rd argument
%rcx	4th argument
%r8	5th argument
%r9	6th argument
%r10, %r11	Caller-saved registers
%rbx, %rbp, %r12-15	Callee-saved registers

Addressing modes

Example source operands to **mov**

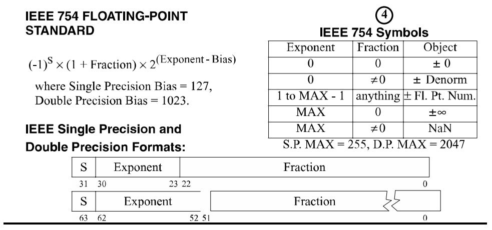
Immediate: mov $0x5, dst
$val
source is constant value

Register: mov %rax, dst
%R, R is register
source in %R

Direct: mov (%rax), dst
source read from Mem[%R]

Indirect displacement:
mov 8(%rax), dst
D(%R), D is displacement
source read from Mem[%R+D]

Indirect scaled-index:
mov 8(%rsp,%rcx,4), dst
D(%RB, %RI, S)
source read from Mem[%RB+D+%RI*S]



IEEE 754 FLOATING-POINT STANDARD

$(-1)^S \times (1 + \text{Fraction}) \times 2^{(\text{Exponent - Bias})}$

where Single Precision Bias = 127, Double Precision Bias = 1023.

4

IEEE 754 Symbols

Exponent	Fraction	Object
0	0	± 0
0	≠0	± Denorm
1 to MAX - 1	anything	± Fl. Pt. Num.
MAX	0	±∞
MAX	≠0	NaN

S.P. MAX = 255, D.P. MAX = 2047

IEEE Single Precision and Double Precision Formats:

S | Exponent | Fraction
31 30 23 22 0
S | Exponent | Fraction
63 62 52 51 0