

# **Super Simple Assembly v3.1**

**ISA: Instruction Set Architecture**

# SSAMv3.1

## Operations

group	group code	operation	Register-Transfer Notation		
(f)low	00	halt	stop computer		
		nop	do nothing		
		ret	$R[SP] \leq R[BP]$ $R[BP] \leq M[R[SP]]$ $R[SP] \leq R[SP] - \text{WordSize-bytes}$ $R[PC] \leq M[R[SP]]$	set stack-ptr base of frame set base-ptr to base of prev frame set stack-ptr to prev next-instr reinstate prev next-instr in PC	
		dump code	debugging operators		
	01	readr rA	debugging operators		
		writr rA	debugging operators		
		writa address	debugging operators		
		stoa (address), rA	$M[\text{address}] \leq R[rA]$		
(t)ransfer	01	stor (rB), rA	$M[R[rB]] \leq R[rA]$		
		stord (rB + ind), rA	$M[R[rB] + \text{ind}] \leq R[rA]$		
		lodi rA, immediate	$R[rA] \leq \text{immediate}$		
		loda rA, (address)	$R[rA] \leq M[\text{address}]$		
		lodr rA, (rB)	$R[rA] \leq M[R[rB]]$		
		lodrd rA, (rB + ind)	$R[rA] \leq M[R[rB] + \text{ind}]$		
(m)anipulate	10	neg rA	$R[AC] \leq -R[rA]$		
		addi rA, immediate	$R[AC] \leq R[rA] + \text{immediate}$		
		addr rA, rB	$R[AC] \leq R[rA] + R[rB]$		
		subi rA, immediate	$R[AC] \leq R[rA] - \text{immediate}$		
		subr rA, rB	$R[AC] \leq R[rA] - R[rB]$		
		mov rA, rB	$R[rA] \leq R[rB]$		
(j)ump	11	jmp address	$R[PC] \leq \text{address}$		
		jmpz address	$R[PC] \leq \text{address}$	if $R[AC] == 0$	
		jmpn address	$R[PC] \leq \text{address}$	if $R[AC]$ is negative	
		call address	$M[R[SP]] \leq R[PC]$ $R[SP] \leq R[SP] + \text{WordSize-byte}$	store current next-instr increment stack-ptr	
			$R[PC] \leq \text{address}$	place new next-instr in PC	
			$M[R[SP]] \leq R[BP]$	store base-ptr	
			$R[BP] \leq R[SP]$	set base-ptr to point at old base-ptr	
			$R[SP] \leq R[SP] + \text{WordSize-byte}$	increment stack-ptr	

# SSAMv3.1

## Fetch-Execute CPU-Memory RTN

```

1. fetch : get instruction @ PC
   : store instruction in IR
   : increment PC
   IR <= M[PC]
   PC <= PC + word-size

```

```

2. decode : determine operation
   : decode instruction as RTN

```

IR	operation	? determine operands ?
----	-----------	------------------------

```

3. execute: perform work of RTN

```

RTN: dest <= src operation src

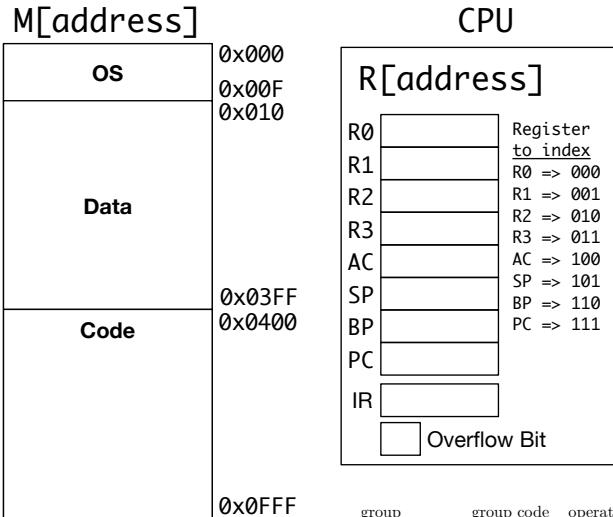
```

4. store : store result

```

RTN: dest <= src operation src

The Fetch-Decide-Execute-Store cycle is this order of sub-operations that obtains the instruction, determines what is happening, do the work, and then store the result.

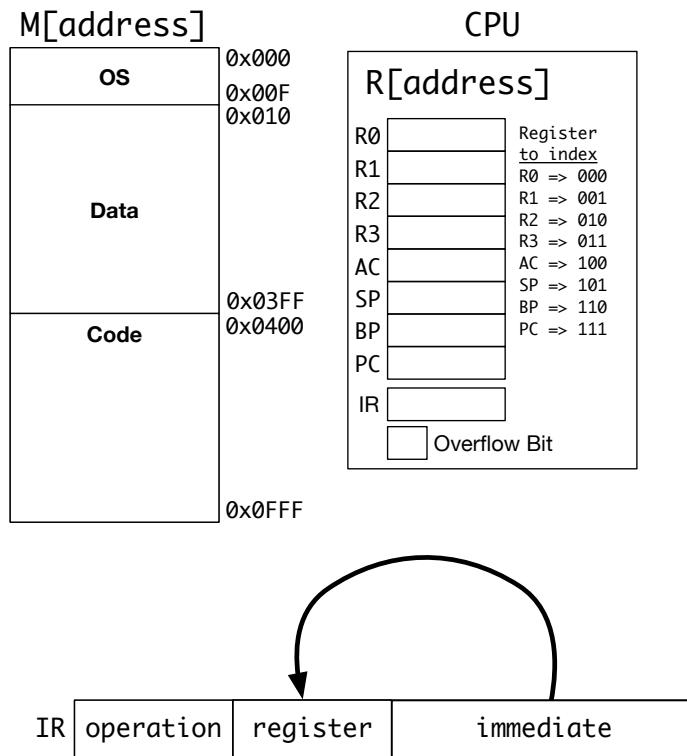


group	group code	operation	Register-Transfer Notation
(f)low	00	halt nop	stop computer do nothing
		ret	R[SP] <= R[BP] R[BP] <= M[R[SP]] R[SP] <= R[SP] - WordSize-bytes R[PC] <= M[R[SP]]
		dump code	set stack-ptr base of frame
		readr rA	set base-ptr to base of prev frame
		writr rA	set stack-ptr to prev next-instr
		writa address	reinstate prev next-instr in PC
(t)ransfer	01	stoa (address), rA stor (rB), rA stord (rB + ind), rA	M[address] <= R[rA] M[R[rB]] <= R[rA] M[R[rB]] + ind <= R[rA]
		lodi rA, immediate lodr rA, (address) lodr rA, (rB) lodrd rA, (rB + ind)	R[rA] <= immediate R[rA] <= M[address] R[rA] <= M[R[rB]] R[rA] <= M[R[rB]] + ind
(m)anipulate	10	neg rA addi rA, immediate addr rA, rB subi rA, immediate subr rA, rB mov rA, rB	R[AC] <= -R[rA] R[AC] <= R[rA] + immediate R[AC] <= R[rA] + R[rB] R[AC] <= R[rA] - immediate R[AC] <= R[rA] - R[rB] R[rA] <= R[rB]
(j)ump	11	jmp address jmpz address jmpn address call address	R[PC] <= address R[PC] <= address R[PC] <= address R[PC] <= address if R[AC] == 0 if R[AC] is negative M[R[SP]] <= R[PC] store current next-instr R[SP] <= R[SP] + WordSize-byte increment stack-ptr R[PC] <= address place new next-instr in PC M[R[SP]] <= R[BP] store base-ptr R[BP] <= R[SP] set base-ptr to point at old base-ptr R[SP] <= R[SP] + WordSize-byte increment stack-ptr

Both memory and registers can be viewed as arrays. The memory array M[address] is composed of bits. The register array R[address] consists of short values, with an indexing that mapped by the register name.

# SSAMv3.1

## Memory Access Modes Immediate Addressing



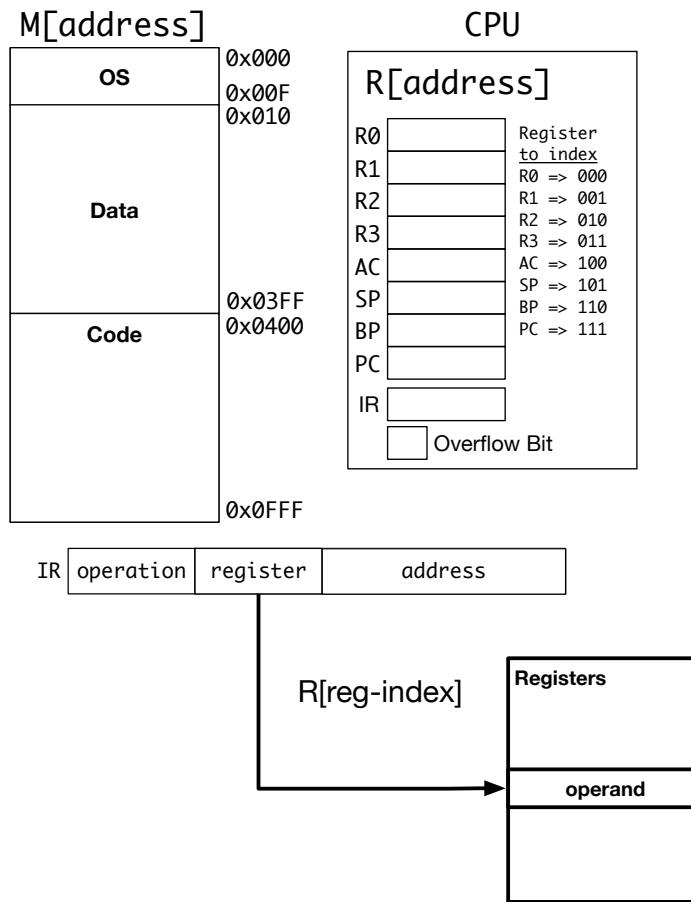
Immediate addressing takes the value from the encoded instruction. Thus, it is something like a literal in a C programming language.

Instructions that use immediate addressing are:  
lodi and stoi

group	group code	operation	Register-Transfer Notation	
(f)low	00	halt	stop computer	
		nop	do nothing	
		ret	$R[SP] \leq R[BP]$ $R[BP] \leq M[R[SP]]$ $R[SP] \leq R[SP] - \text{WordSize-bytes}$ $R[PC] \leq M[R[SP]]$	set stack-ptr base of frame set base-ptr to base of prev frame set stack-ptr to prev next-instr reinstate prev next-instr in PC
		dump code	debugging operators	
		readr rA	debugging operators	
		writr rA	debugging operators	
		writa address	debugging operators	
(t)ransfer	01	stoa (address), rA	$M[\text{address}] \leq R[rA]$	
		stor (rB), rA	$M[R[rB]] \leq R[rA]$	
		stord (rB + ind), rA	$M[R[rB] + \text{ind}] \leq R[rA]$	
		lodi rA, immediate	$R[rA] \leq \text{immediate}$	
		loda rA, (address)	$R[rA] \leq M[\text{address}]$	
		lodr rA, (rB)	$R[rA] \leq M[R[rB]]$	
		lodrd rA, (rB + ind)	$R[rA] \leq M[R[rB] + \text{ind}]$	
(m)anipulate	10	neg rA	$R[AC] \leq -R[rA]$	
		addi rA, immediate	$R[AC] \leq R[rA] + \text{immediate}$	
		addr rA, rB	$R[AC] \leq R[rA] + R[rB]$	
		subi rA, immediate	$R[AC] \leq R[rA] - \text{immediate}$	
		subr rA, rB	$R[AC] \leq R[rA] - R[rB]$	
		mov rA, rB	$R[rA] \leq R[rB]$	
(j)ump	11	jmp address	$R[PC] \leq \text{address}$	
		jmpz address	$R[PC] \leq \text{address}$	
		jmpn address	$R[PC] \leq \text{address}$	
		call address	$M[R[SP]] \leq R[PC]$ $R[SP] \leq R[SP] + \text{WordSize-byte}$ $R[PC] \leq \text{address}$	if $R[AC] == 0$ if $R[AC]$ is negative store current next-instr increment stack-ptr place new next-instr in PC
			$M[R[SP]] \leq R[BP]$ $R[BP] \leq R[SP]$ $R[SP] \leq R[SP] + \text{WordSize-byte}$	store base-ptr set base-ptr to point at old base-ptr increment stack-ptr

# SSAMv3.1

## Memory Access Modes Register Addressing



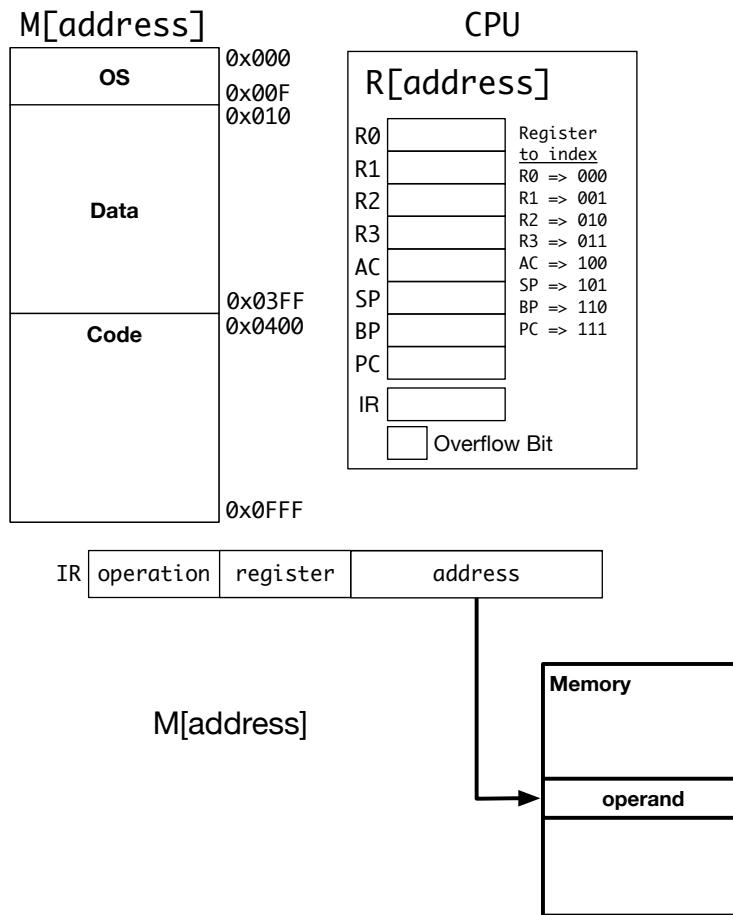
Register addressing indicate the register that will be used by the instruction, which is encoded in the instruction.

Instructions that use immediate addressing are:  
**mov** and **addr**

group	group code	operation	Register-Transfer Notation	
(f)low	00	halt	stop computer	
		nop	do nothing	
		ret	$R[SP] \leq R[BP]$ $R[BP] \leq M[R[SP]]$ $R[SP] \leq R[SP] - \text{WordSize-bytes}$ $R[PC] \leq M[R[SP]]$	set stack-ptr base of frame set base-ptr to base of prev frame set stack-ptr to prev next-instr reinstate prev next-instr in PC
		dump code	debugging operators	
		readr rA	debugging operators	
		writr rA	debugging operators	
		writa address	debugging operators	
(t)ransfer	01	stoa (address), rA	$M[\text{address}] \leq R[rA]$	
		stor (rB), rA	$M[R[rB]] \leq R[rA]$	
		stord (rB + ind), rA	$M[R[rB] + \text{ind}] \leq R[rA]$	
		lodi rA, immediate	$R[rA] \leq \text{immediate}$	
		loda rA, (address)	$R[rA] \leq M[\text{address}]$	
		lodr rA, (rB)	$R[rA] \leq M[R[rB]]$	
		lodrd rA, (rB + ind)	$R[rA] \leq M[R[rB] + \text{ind}]$	
(m)anipulate	10	neg rA	$R[AC] \leq -R[rA]$	
		addi rA, immediate	$R[AC] \leq R[rA] + \text{immediate}$	
		addr rA, rB	$R[AC] \leq R[rA] + R[rB]$	
		subi rA, immediate	$R[AC] \leq R[rA] - \text{immediate}$	
		subr rA, rB	$R[AC] \leq R[rA] - R[rB]$	
		mov rA, rB	$R[rA] \leq R[rB]$	
(j)ump	11	jmp address	$R[PC] \leq \text{address}$	
		jmpz address	$R[PC] \leq \text{address}$	
		jmpn address	$R[PC] \leq \text{address}$	
		call address	$M[R[SP]] \leq R[PC]$ $R[SP] \leq R[SP] + \text{WordSize-byte}$ $R[PC] \leq \text{address}$	if $R[AC] == 0$ store current next-instr increment stack-ptr place new next-instr in PC
			$M[R[SP]] \leq R[BP]$ $R[BP] \leq R[SP]$ $R[SP] \leq R[SP] + \text{WordSize-byte}$	store base-ptr set base-ptr to point at old base-ptr increment stack-ptr

# SSAMv3.1

## Memory Access Modes Direct Addressing



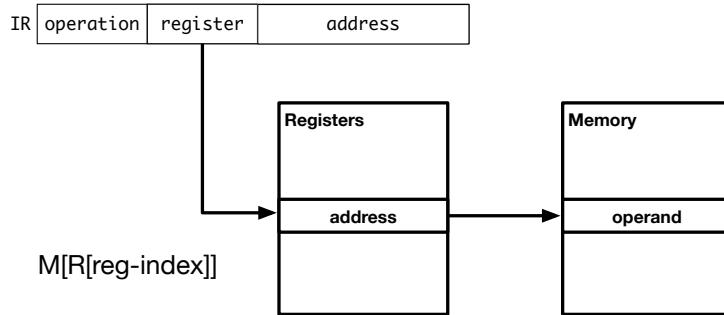
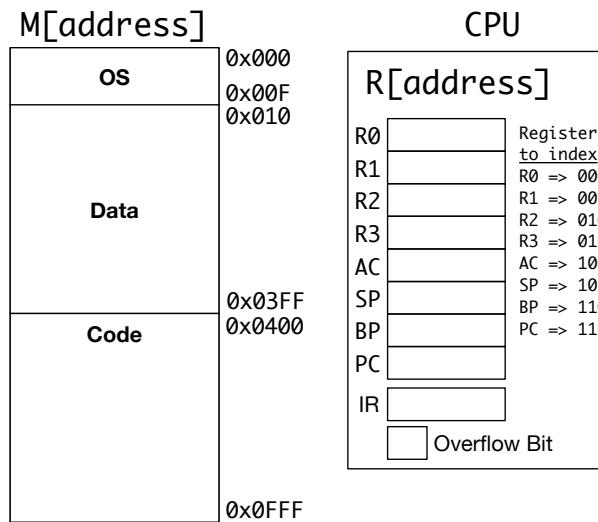
Direct addressing will take the memory address from the instruction.

Instructions that use immediate addressing are:  
loda and stoa

group	group code	operation	Register-Transfer Notation	
(f)low	00	halt nop ret	stop computer do nothing R[SP] <= R[BP] R[BP] <= M[R[SP]] R[SP] <= R[SP] - WordSize-bytes R[PC] <= M[R[SP]]	
		dump code readr rA writr rA writa address	set stack-ptr base of frame set base-ptr to base of prev frame set stack-ptr to prev next-instr reinstate prev next-instr in PC	
(t)ransfer	01	stoa (address), rA stor (rB), rA stord (rB + ind), rA lodl rA, immediate loda rA, (address) lodr rA, (rB) lodrd rA, (rB + ind)	M[address] <= R[rA] M[R[rB]] <= R[rA] M[R[rB]] + ind <= R[rA] R[rA] <= immediate R[rA] <= M[address] R[rA] <= M[R[rB]] R[rA] <= M[R[rB]] + ind	
(m)anipulate	10	neg rA addi rA, immediate addr rA, rB subi rA, immediate subr rA, rB mov rA, rB	R[AC] <= -R[rA] R[AC] <= R[rA] + immediate R[AC] <= R[rA] + R[rB] R[AC] <= R[rA] - immediate R[AC] <= R[rA] - R[rB] R[rA] <= R[rB]	
(j)ump	11	jmp address jmpz address jmpn address call address	R[PC] <= address R[PC] <= address R[PC] <= address M[R[SP]] <= R[PC] R[SP] <= R[SP] + WordSize-byte R[PC] <= address M[R[SP]] <= R[BP] R[BP] <= R[SP] R[SP] <= R[SP] + WordSize-byte	if R[AC] == 0 if R[AC] is negative store current next-instr increment stack-ptr place new next-instr in PC store base-ptr set base-ptr to point at old base-ptr increment stack-ptr

# SSAMv3.1

## Memory Access Modes Register Indirect Addressing



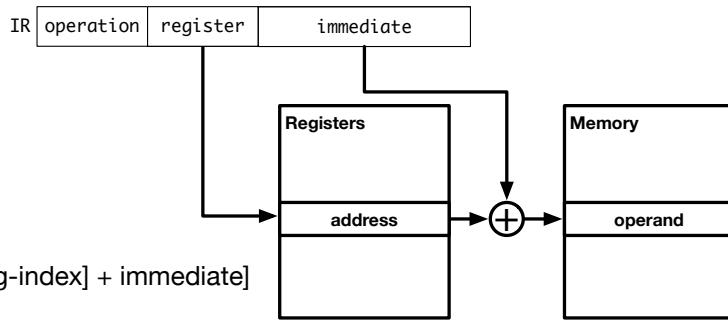
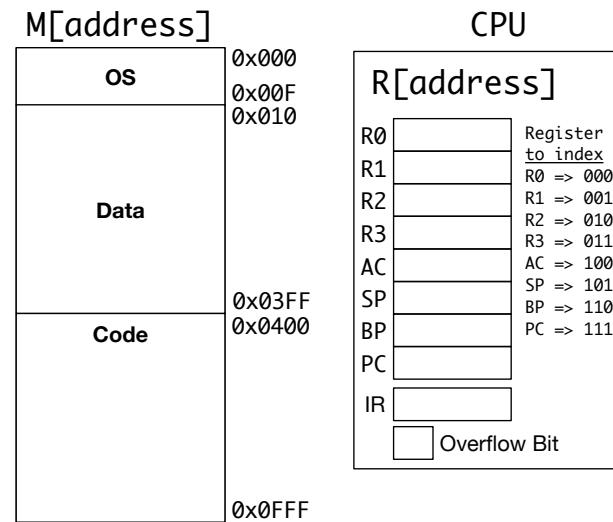
Register Indirect addressing will take the a memory address that is stored in the register. Thus, the register is accessed and that value is the used to index the memory.

Instructions that use immediate addressing are:  
lodr and stor

group	group code	operation	Register-Transfer Notation
(f)low	00	halt nop ret	stop computer do nothing R[SP] <= R[BP] R[BP] <= M[R[SP]] R[SP] <= R[SP] - WordSize-bytes R[PC] <= M[R[SP]]
		dump code readr rA writr rA writa address	set stack-ptr base of frame set base-ptr to base of prev frame set stack-ptr to prev next-instr reinstate prev next-instr in PC
(t)ransfer	01	stoa (address), rA stor (rB), rA stord (rB + ind), rA lodri rA, immediate loda rA, (address) lodr rA, (rB) lodrd rA, (rB + ind)	M[address] <= R[rA] M[R[rB]] <= R[rA] M[R[rB]] + ind <= R[rA] R[rA] <= immediate R[rA] <= M[address] R[rA] <= M[R[rB]] R[rA] <= M[R[rB]] + ind
(m)anipulate	10	neg rA addi rA, immediate addr rA, rB subi rA, immediate subr rA, rB mov rA, rB	R[AC] <= -R[rA] R[AC] <= R[rA] + immediate R[AC] <= R[rA] + R[rB] R[AC] <= R[rA] - immediate R[AC] <= R[rA] - R[rB] R[rA] <= R[rB]
(j)ump	11	jmp address jmpz address jmpn address call address	R[PC] <= address R[PC] <= address R[PC] <= address M[R[SP]] <= R[PC] R[SP] <= R[SP] + WordSize-byte R[PC] <= address
			if R[AC] == 0 if R[AC] is negative store current next-instr increment stack-ptr place new next-instr in PC
			M[R[SP]] <= R[BP] R[BP] <= R[SP] R[SP] <= R[SP] + WordSize-byte
			store base-ptr set base-ptr to point at old base-ptr increment stack-ptr

# SSAMv3.1

## Memory Access Modes Register Displacement Addressing



Register Displacement addressing will take the a memory address that is stored in the register, then use an index to determine an offset. This is useful for implementing arrays.

Instructions that use immediate addressing are:  
lodrd and stord

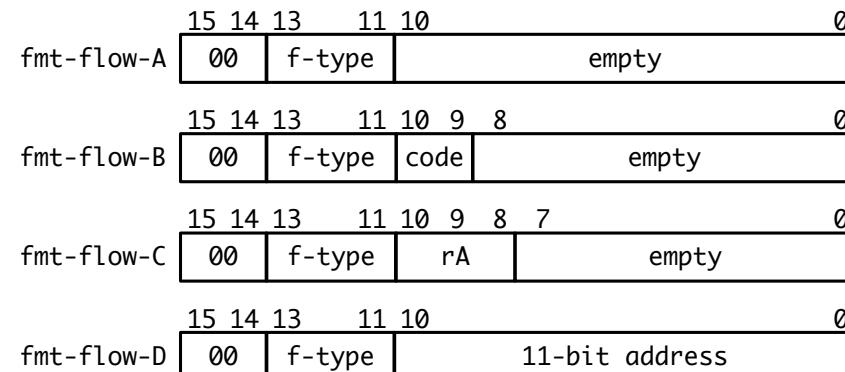
group	group code	operation	Register-Transfer Notation
(f)low	00	halt nop ret	stop computer do nothing R[SP] <= R[BP] R[BP] <= M[R[SP]] R[SP] <= R[SP] - WordSize-bytes R[PC] <= M[R[SP]]
		dump code readr rA writr rA writa address	debugging operators debugging operators debugging operators debugging operators
(t)transfer	01	stoa (address), rA stor (rB), rA stord (rB + ind), rA lodi rA, immediate loda rA, (address) lodr rA, (rB) lodrd rA, (rB + ind)	M[address] <= R[rA] M[R[rB]] <= R[rA] M[R[rB] + ind] <= R[rA] R[rA] <= immediate R[rA] <= M[address] R[rA] <= M[R[rB]] R[rA] <= M[R[rB] + ind]
(m)anipulate	10	neg rA addi rA, immediate addr rA, rB subi rA, immediate subr rA, rB mov rA, rB	R[AC] <= -R[rA] R[AC] <= R[rA] + immediate R[AC] <= R[rA] + R[rB] R[AC] <= R[rA] - immediate R[AC] <= R[rA] - R[rB] R[rA] <= R[rB]
(j)ump	11	jmp address jmpz address jmpn address call address	R[PC] <= address R[PC] <= address R[PC] <= address M[R[SP]] <= R[PC] R[SP] <= R[SP] + WordSize-byte R[PC] <= address
			if R[AC] == 0 if R[AC] is negative store current next-instr increment stack-ptr place new next-instr in PC
			M[R[SP]] <= R[BP] R[BP] <= R[SP] R[SP] <= R[SP] + WordSize-byte
			store base-ptr set base-ptr to point at old base-ptr increment stack-ptr

# SSAMv3.1

## Flow Operations (f) — 00

group	group code	operation	Register-Transfer Notation
(f)low	00	halt	stop computer
		nop	do nothing
		ret	$R[SP] \leq R[BP]$ set stack-ptr base of frame $R[BP] \leq M[R[SP]]$ set base-ptr to base of prev frame $R[SP] \leq R[SP] - \text{WordSize-bytes}$ set stack-ptr to prev next-instr $R[PC] \leq M[R[SP]]$ reinstate prev next-instr in PC
		dump code	debugging operators
		readr rA	debugging operators
		writr rA	debugging operators
		writa address	debugging operators

f-type	operation	format
000	halt	fmt-flow-A
001	nop	fmt-flow-A
010	ret	fmt-flow-A
011		
100	dump code	fmt-flow-B
101	read rA	fmt-flow-C
110	writr rA	fmt-flow-C
111	write (address)	fmt-flow-D



code	action	rX	register
00	all	000	R0
01	reg	001	R1
10	mem	010	R2
11		011	R3
		100	AC
		101	SP
		110	BP
		111	PC

Notes: 11-bit address is stored in unsigned.

# SSAMv3.1

## Transfer Operations (t) — 01

group	group code	operation	Register-Transfer Notation					
(t)transfer	01	stoa (address), rA	M[address]	<= R[rA]				
		stor (rB), rA	M[R[rB]]	<= R[rA]				
		stord (rB + ind), rA	M[R[rB] + ind]	<= R[rA]				
		lodi rA, immediate	R[rA]	<= immediate				
		loda rA, (address)	R[rA]	<= M[address]				
		lodr rA, (rB)	R[rA]	<= M[R[rB]]				
		lodrd rA, (rB + ind)	R[rA]	<= M[R[rB] + ind]				

t-type(tt)_mod_operation		
0	00	lodi
0	01	loda
0	10	lodr
0	11	lodrd
1	00	stoa
1	01	stor
1	10	stord
1	11	

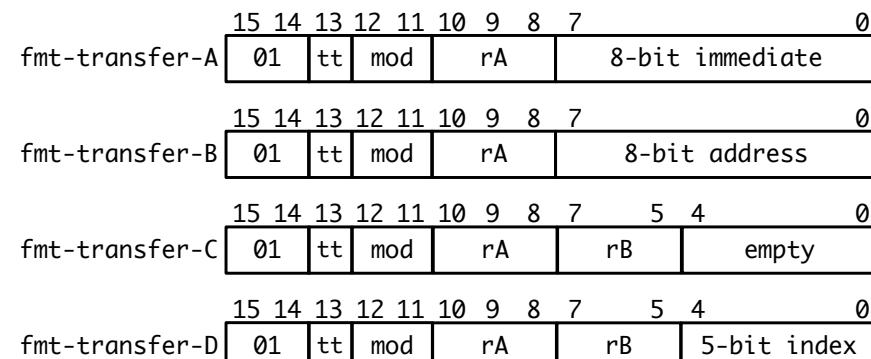
rX	register
000	R0
001	R1
010	R2
011	R3
100	AC
101	SP
110	BP
111	PC

Notes:

8-bit immediate is stored in 2's Complement.

5-bit index is stored in 2's Complement.

8-bit address is stored in unsigned.

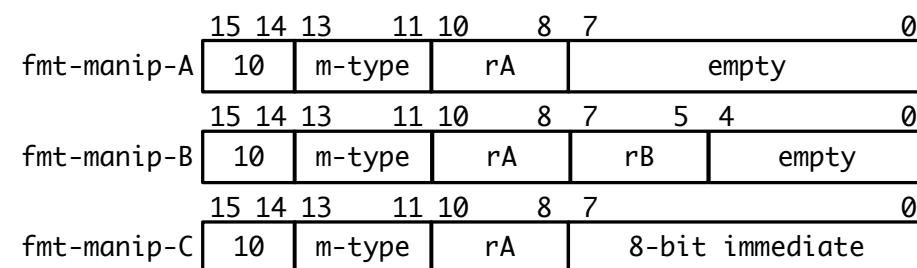


# SSAMv3.1

## Manipulate Operations (m) – 10

group	group code	operation	Register-Transfer Notation
(m)anipulate	10	neg rA	$R[AC] \leq -R[rA]$
		addi rA, immediate	$R[AC] \leq R[rA] + \text{immediate}$
		addr rA, rB	$R[AC] \leq R[rA] + R[rB]$
		subi rA, immediate	$R[AC] \leq R[rA] - \text{immediate}$
		subr rA, rB	$R[AC] \leq R[rA] - R[rB]$
		mov rA, rB	$R[rA] \leq R[rB]$

m-type	operation	format
000	neg	fmt-manip-A
001	addr	fmt-manip-B
010	addi	fmt-manip-C
011	subr	fmt-manip-B
100	subi	fmt-manip-C
101		
110		
111	mov	fmt-manip-B



rX	register
000	R0
001	R1
010	R2
011	R3
100	AC
101	SP
110	BP
111	PC

Notes: 8-bit immediate is stored in 2's Complement.

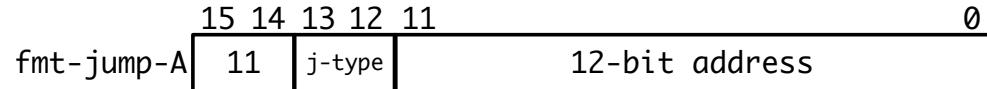
# SSAMv3.1

## Jump Operations (j) – 11

group	group	code	operation	Register-Transfer Notation	
(j)ump	11	jmp address	R[PC]	<= address	
		jmpz address	R[PC]	<= address	if R[AC] == 0
		jmpn address	R[PC]	<= address	if R[AC] is negative
		call address	M[R[SP]]	<= R[PC]	store current next-instr
			R[SP]	<= R[SP] + WordSize-byte	increment stack-ptr
			R[PC]	<= address	place new next-instr in PC
		M[R[SP]] <= R[BP]	M[R[SP]]	<= R[BP]	store base-ptr
			R[BP]	<= R[SP]	set base-ptr to point at old base-ptr
			R[SP]	<= R[SP] + WordSize-byte	increment stack-ptr

### j-type operation format

00	jmp	fmt-jump-A
01	jmpz	fmt-jump-A
10	jmpn	fmt-jump-A
11	call	fmt-jump-A



Notes: 8-bit address is stored in unsigned.

# SSAMv3.1

## Combined Formats

fmt-flow-A	15 14 13    11 10	0
	00   f-type	empty
fmt-flow-B	15 14 13    11 10 9 8	0
	00   f-type   code	empty
fmt-flow-C	15 14 13    11 10 9 8 7	0
	00   f-type   rA	empty
fmt-flow-D	15 14 13    11 10	0
	00   f-type	11-bit address
fmt-transfer-A	15 14 13 12 11 10 9 8 7	0
	01   tt   mod   rA	8-bit immediate
fmt-transfer-B	15 14 13 12 11 10 9 8 7	0
	01   tt   mod   rA	8-bit address
fmt-transfer-C	15 14 13 12 11 10 9 8 7    5 4	0
	01   tt   mod   rA   rB	empty
fmt-transfer-D	15 14 13 12 11 10 9 8 7    5 4	0
	01   tt   mod   rA   rB	5-bit index
fmt-manip-A	15 14 13    11 10    8 7	0
	10   m-type   rA	empty
fmt-manip-B	15 14 13    11 10    8 7    5 4	0
	10   m-type   rA   rB	empty
fmt-manip-C	15 14 13    11 10    8 7	0
	10   m-type   rA	8-bit immediate
fmt-jump-A	15 14 13 12 11	0
	11   j-type	12-bit address

Note: 11-bit address is stored in unsigned.

Note: 8-bit immediate is stored in 2's Complement.

Note: 8-bit address is stored in unsigned.

Note: 5-bit index is stored in 2's Complement.

Note: 8-bit immediate is stored in 2's Complement.

Note: 8-bit address is stored in unsigned.

## 4 Command types 12 Format types

## 3 Addressing types

- 8-bit unsigned
- 11-bit unsigned
- 12-bit unsigned

## 2 Immediate type

- 8-bit 2's Complement
- 5-bit 2's Complement

# **SSAMv3.1**

## assembly process

```
.file exampleCode/inclass.asm
.format SSAM-label-assembly
0x0610    loda R1, (0x012)
0x0612    lodl R2, 0x0

loop:
0x0614    subi R1, 0x1
0x0616    mov R1, AC
0x0618    addi R2, 0x1
0x061a    mov R2, AC
0x061c    subi R2, 0x3
0x061e    jmpn loop
0x0620    stoa (0x012), R1
0x0622    halt
```

```
.file exampleCode/inclass.asm
.format SSAM-address-assembly
0x0610    >16>>0100100100010010<<< 0x4912 loda R1, (0x012)
0x0612    >16>>0100001000000000<<< 0x4200 lodl R2, 0x0

loop:
0x0614    >16>>1010000100000001<<< 0xA101 subi R1, 0x1
0x0616    >16>>1011100110000000<<< 0xB980 mov R1, AC
0x0618    >16>>1001001000000001<<< 0x9201 addi R2, 0x1
0x061a    >16>>1011101010000000<<< 0xBA80 mov R2, AC
0x061c    >16>>1010001000000011<<< 0xA203 subi R2, 0x3
0x061e    >16>>1110011000010100<<< 0xE614 jmpn 0x0614
0x0620    >16>>0110000100010010<<< 0x6112 stoa (0x012), R1
0x0622    >16>>0000000000000000<<< 0x0000 halt
```

# SSAMv3.1

assembly process  
executable image

```
pfaffmaj@Jeffreys-MacBook-Pro simulator_base2 % hexdump -C binary.bin
00000000  ba ab 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....
00000010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....
*
00000610  49 12 42 00 a1 01 b9 80  92 01 ba 80 a2 03 e6 14 |I.B.....
00000620  61 12 00 00
00000624
```

```
.file exampleCode/inclass.asm
.format SSAM-address-assembly
0x0610 >16>>0100100100010010<<< 0x4912 loda R1, (0x012)
0x0612 >16>>0100001000000000<<< 0x4200 lodl R2, 0x0

loop:
0x0614 >16>>1010000100000001<<< 0xA101 subi R1, 0x1
0x0616 >16>>1011100110000000<<< 0xB980 mov R1, AC
0x0618 >16>>1001001000000001<<< 0x9201 addi R2, 0x1
0x061a >16>>1011101010000000<<< 0xBA80 mov R2, AC
0x061c >16>>1010001000000011<<< 0xA203 subi R2, 0x3
0x061e >16>>1110011000010100<<< 0xE614 jmpn 0x0614
0x0620 >16>>0110000100010010<<< 0x6112 stoa (0x012), R1
0x0622 >16>>0000000000000000<<< 0x0000 halt
```

# SSAMv3.1

## Combined Formats

### Where to add Push and Pop?

fmt-flow-A	15 14 13    11 10	0
	00   f-type	empty
fmt-flow-B	15 14 13    11 10 9 8	0
	00   f-type   code	empty
fmt-flow-C	15 14 13    11 10 9 8 7	0
	00   f-type   rA	empty
fmt-flow-D	15 14 13    11 10	0
	00   f-type	11-bit address
fmt-transfer-A	15 14 13 12 11 10 9 8 7	0
	01   tt   mod   rA	8-bit immediate
fmt-transfer-B	15 14 13 12 11 10 9 8 7	0
	01   tt   mod   rA	8-bit address
fmt-transfer-C	15 14 13 12 11 10 9 8 7 5 4	0
	01   tt   mod   rA   rB	empty
fmt-transfer-D	15 14 13 12 11 10 9 8 7 5 4	0
	01   tt   mod   rA   rB	5-bit index
fmt-manip-A	15 14 13    11 10	0
	10   m-type   rA	empty
fmt-manip-B	15 14 13    11 10    8 7 5 4	0
	10   m-type   rA   rB	empty
fmt-manip-C	15 14 13    11 10    8 7	0
	10   m-type   rA	8-bit immediate
fmt-jump-A	15 14 13 12 11	0
	11   j-type	12-bit address

Note: 11-bit address is stored in unsigned.

Note: 8-bit immediate is stored in 2's Complement.

Note: 8-bit address is stored in unsigned.

Note: 5-bit index is stored in 2's Complement.

Note: 8-bit immediate is stored in 2's Complement.

Note: 8-bit address is stored in unsigned.

- 4 Command types
- 12 Format types
- 8 Register Count

- 3 Addressing types
  - 8-bit unsigned
  - 11-bit unsigned
  - 12-bit unsigned
- 2 Immediate type
  - 8-bit 2's Complement
  - 5-bit 2's Complement

# SSAMv3.1

## Combined Formats

### Where to add Push and Pop?

<u>f-type</u>	<u>operation</u>	<u>format</u>	<u>t-type(tt)</u>	<u>mod</u>	<u>operation</u>	<u>format</u>
000	halt	fmt-flow-A	0	00	lodi	fmt-transfer-A
001	nop	fmt-flow-A	0	01	loda	fmt-transfer-B
010	ret	fmt-flow-A	0	10	lodr	fmt-transfer-C
011			0	11	lodrd	fmt-transfer-D
100	dump code	fmt-flow-B	1	00	stoa	fmt-transfer-B
101	read rA	fmt-flow-C	1	01	stor	fmt-transfer-C
110	writr rA	fmt-flow-C	1	10	stord	fmt-transfer-D
111	write (address)	fmt-flow-D	1	11		

<u>j-type</u>	<u>operation</u>	<u>format</u>	<u>m-type</u>	<u>operation</u>	<u>format</u>
00	jmp	fmt-jump-A	000	neg	fmt-manip-A
01	jmpz	fmt-jump-A	001	addr	fmt-manip-B
10	jmpn	fmt-jump-A	010	addi	fmt-manip-C
11	call	fmt-jump-A	011	subr	fmt-manip-B
			100	subi	fmt-manip-C
			101		
			110		
			111	mov	fmt-manip-B

- 4 Command types
- 12 Format types
- 8 Register Count
- 3 Addressing types
  - 8-bit unsigned
  - 11-bit unsigned
  - 12-bit unsigned
- 2 Immediate type
  - 8-bit 2's Complement
  - 5-bit 2's Complement

# **SSAMv3.1**

**Variable Width Instructions**

Where to add Push and Pop?

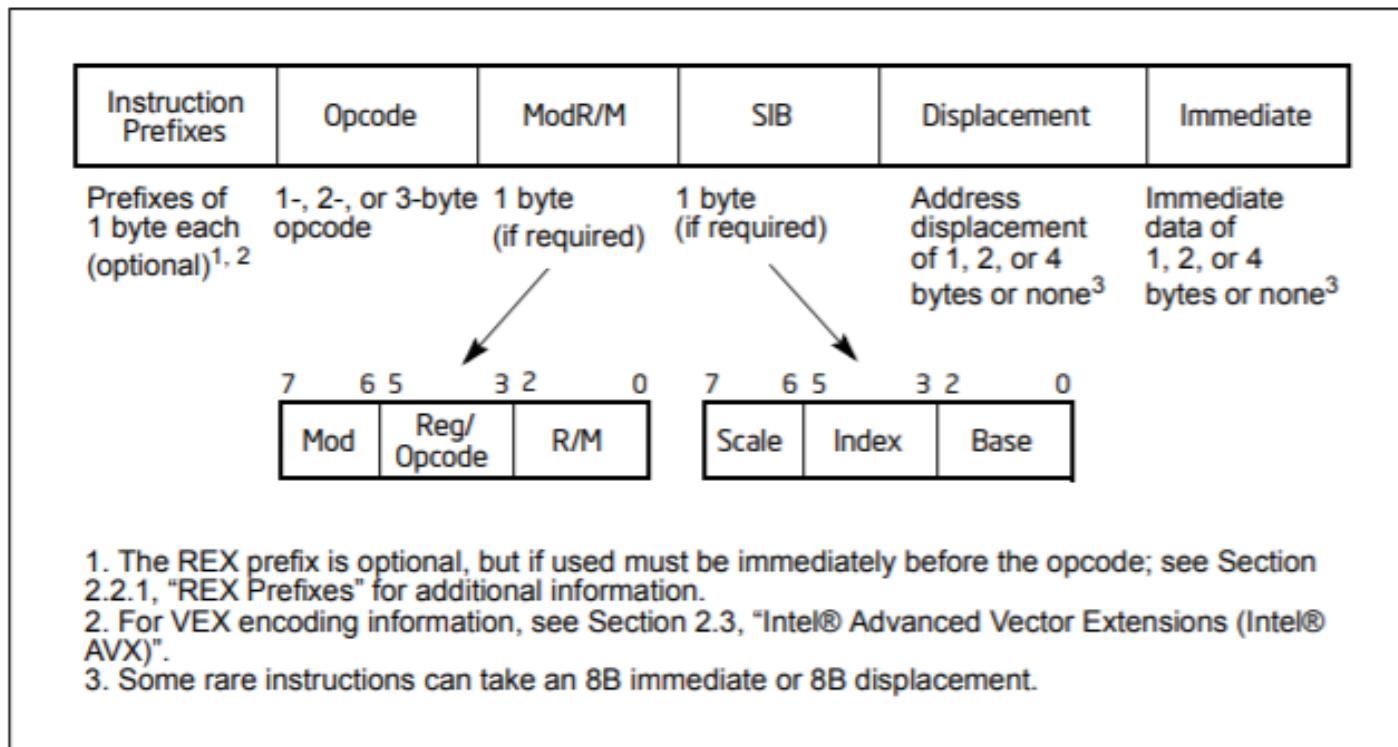
4 Command types  
64 Commands per type  
16 Register Count (or 256)

1 Addressing types  
1 Immediate type

Gives more space, but  
not always simple.

# SSAMv3.1

## Variable Width Instructions



Gives more space, but not always simple.

Figure 2-1. Intel 64 and IA-32 Architectures Instruction Format