Computer Architecture Simulators for Different Instruction Formats

Xuejun Liang
Department of Computer Science
California State University – Stanislaus
Turlock, CA 95382, USA

Introduction

- Assembly language programming and writing, using and modifying processor simulators are major hands-on assignment categories in an undergraduate computer architecture and organization course.
- There are many computer architectures with different instruction formats such as stack-based, accumulator-based, two-address, or three-address machine.
- It is certainly desirable to have various simple simulators, each for one major computer processor architecture, so that students can program and compare these processors.

Simulated Instruction Sets

- Stack-based (Zero address) Machine
- Accumulator-based (one address) Machine
- Two address Machine
  - Memory-to-Memory
  - Memory-to-Register
  - Register-to-Register
- Three address Machine
  - Memory-to-Memory
  - Register-to-Register

Program Structure and Syntax (1)

- Every program contains three sections and separated by END
  - Data (optional)
  - Code
  - Input (optional)
- Data (Declarations)
  - One variable definition per line
  - **ID** (identifier) is the variable name.
  - **Type** is a positive integer
    - **Type** = 1, ID is a scalar variable
    - **Type** > 1, ID is an array variable
  - **Value** is up to **Type** initial integers of **ID.** If less than **Type** initial values are provided, default initial values are used.

[Data]
END
Code
END
[Input]

ID Type [Value]

[Label:] Instruction

Number (integer)

Program Structure and Syntax (2)

- Code (Instructions)
  - One instruction per line
  - **Label** is optional. It must be followed by ':' immediately. There is no space between Label and ':'.
  - **Instruction** is any instruction, including pseudo-instruction.
- Input:
  - One input value per line.
  - **Number** is any integer.
- Comments:
  - Any text starting from // to the end of the line will be considered as comments

[Data]
END
Code
END
[Input]

ID Type [Value]

[Label:] Instruction

Number (integer)

Example 1: Add Two Numbers

Accumulator-Based

```
//Program A
//Declaration
Num1    1       //Variable holding the first number
Num2    1       //Variable holding the second number
Sum     1       //Variable the sum
END
//Code
    READ        //Read the first number, AC = 23
    PUT Num1    //Store the first number in Num1
    READ        //Read the second number, AC = 48
    PUT Num2    //Store the second number in Num2
    ADD Num1    //Add the first number, AC = 48+23
    PUT Sum     //Store sum at address Sum
    PRNT        //Print the Sum
    STOP        //Terminate program
END
//User input
23              //The first number to add
48              //The first number to add
```

Example 1: Add Two Numbers **Accumulator-Based**

```
//Program B
//Declaration
Num1    1   23  //The first number to add
Num2    1   48  //The second number to add
Sum     1       //The sum
END
//Code
    GET Num1    //Get the first number, AC = 23
    ADD Num2    //Add the second number, AC = 23+48
    PUT Sum     //Store sum at address Sum
    PRNT        //Print the Sum
    STOP        //Terminate program
END
//No user input
```

Example 2: Compute Z = (X+Y)*(W-Y)

Stack machine

```
//Stack machine code

//Compute expression (X+Y)*(W-Y)
//Save to Z
//Print Z

//Declarations
X 1 2   //Variable X = 2
Y 1 3   //Variable Y = 3
W 1 6   //Variable W = 6
Z 1     //Variable Z = ?
END
```

Postfix notation: XY+WY-*

```
//Instructions
PUSH X      //X
PUSH Y      //Y
ADD         //X+Y
PUSH W      //W
PUSH Y      //Y
SUB         //W-Y
MUL         //(X+Y)*(W-Y)
POP Z       //Z = (X+Y)*(W-Y)

PUSH Z      //Z
PRNT        //Print Z
STOP        //Terminate
END
```

Example 2: Compute Z = (X+Y)*(W-Y)

Accumulator

```
//Stack machine code

//Compute expression (X+Y)*(W-Y)
//Save to Z
//Print Z

//Declarations
X 1 2   //Variable X = 2
Y 1 3   //Variable Y = 3
W 1 6   //Variable W = 6
Z 1     //Variable Z = ?
END
```

```
//Instructions
GET X       //AC = X
ADD Y       //AC = X+Y
PUT Z       //Z  = X+Y
GET W       //AC = W
SUB Y       //AC = W-Y
MUL Z       //AC = (X+Y)*(W-Y)
PUT Z       //Z  = (X+Y)*(W-Y)

PRNT        //Print Z
STOP        //Terminate
END
```

Example 2: Compute Z = (X+Y)*(W-Y)

Two-address memory-to-memory

```
//Two-address memory-to-memory

//Compute expression (X+Y)*(W-Y)
//Save to Z
//Print Z

//Declarations
X 1 2   //Variable X = 2
Y 1 3   //Variable Y = 3
W 1 6   //Variable W = 6
Z 1     //Variable Z = ?
T 1     //Variable Z = ?
END
```

```
//Instructions
LI    Z 0    //Z = 0
ADD   Z X    //Z = Z+X = X
ADD   Z Y    //Z = Z+Y = X+Y
LI    T 0    //T = 0
ADD   T W    //T = T+W = W
SUB   T Y    //T = T-Y = W-Y
MUL   Z T    //Z = Z*T = (X+Y)*(W-Y)

LI    OUTPUT 0
ADD   OUTPUT Z  //OUTPUT = Z
PRNT            //Print OUTPUT (Z)
STOP            //Terminate
END
```

Example 2: Compute Z = (X+Y)*(W-Y)

Two-address memory-to-register

```
//Two-address Memory-to-register

//Compute expression (X+Y)*(W-Y)
//Save to Z
//Print Z

//Declarations
X 1 2   //Variable X = 2
Y 1 3   //Variable Y = 3
W 1 6   //Variable W = 6
Z 1     //Variable Z = ?
END
```

```
//Instructions
GET $a0 X       //$a0 = X
ADD $a0 Y       //$a0 = X+Y
GET $t2 W       //$t2 = W
SUB $t2 Y       //$t2 = W-Y
MUL $a0 $t2     //$a0 = (X+Y)*(W-Y)
PUT $a0 Z       //Z   = (X+Y)*(W-Y)

PRNT            //Print $a0 (Z)
STOP            //Terminate
END
```

Example 2: Compute Z = (X+Y)*(W-Y)

Two-address register-to-register

```
//Two-address Register-to-register

//Compute expression (X+Y)*(W-Y)
//Save to Z
//Print Z

//Declarations
X 1 2    //Variable X = 2
Y 1 3    //Variable Y = 3
W 1 6    //Variable W = 6
Z 1      //Variable Z = ?
END
```

```
//Instructions
LA  $s0 X        //$s0 = address of X
GET $a0 $s0      //$a0 = X
LA  $s0 Y        //$s0 = address of Y
GET $t2 $s0      //$t2 = Y
LA  $s0 W        //$s0 = address of W
GET $t3 $s0      //$t3 = W

ADD $a0 $t2      //$a0 = X+Y
SUB $t3 $t2      //$t3 = W-Y
MUL $a0 $t3      //$a0 = (X+Y)*(W-Y)

PRNT             //Print $a0 (Z)
STOP             //Terminate
END
```

Example 2: Compute Z = (X+Y)*(W-Y)

Three-address Memory-to-memory

```
//Three-address register-to-register

//Compute expression (X+Y)*(W-Y)
//Save to Z
//Print Z

//Declarations
X 1 2    //Variable X = 2
Y 1 3    //Variable Y = 3
W 1 6    //Variable W = 6
Z 1      //Variable Z = ?
T 1      //Variable T = ?
END
```

```
//Instructions
ADD Z X Y          //Z = X+Y
SUB T W Y          //T = W-Y
MUL Z Z T          //Z = (X+Y)*(W-Y)

ADD OUTPUT ZERO Z    //OUTPUT = Z
PRNT                 //Print Z
STOP//Terminate
END
```

Example 2: Compute Z = (X+Y)*(W-Y)

Three-address register-to-register

```
//Three-address register-to-register

//Compute expression (X+Y)*(W-Y)
//Save to Z
//Print Z

//Declarations
X 1 2    //Variable X = 2
Y 1 3    //Variable Y = 3
W 1 6    //Variable W = 6
Z 1      //Variable Z = ?
END
```

```
//Instructions
GET $t1 $zero X      //$t1 = X
GET $t2 $zero Y      //$t2 = Y
GET $t3 $zero W      //$t3 = W

ADD $t4 $t1 $t2      //$t4 = X+Y
SUB $t5 $t3 $t2      //$t5 = W-Y
MUL $a0 $t4 $t5      //$a0 = (X+Y)*(W-Y)

PUT $a0 $zero Z      //Z = (X+Y)*(W-Y)

PRNT                 //Print Z
STOP                 //Terminate
END
```

Instruction Set (1)

Stack-based Machine

Imm	16-bit 2's complement
PC	Program counter
Var	Variable
Lab	Label
M[A]	Memory content of variable A
SP	Reserved location, Stack pointer
FP	Reserved location, Frame pointer

op	Instruction	Explanation
0	ADD	Pop the top two locations, add, and push the result
1	SUB	Pop subtrahend and minuend, subtract, and push the result
2	MUL	Pop the multiplicand and multiplier, multiply, and push the result
3	DIV	Pop the dividend and divisor, divide, and push the quotient
4	REM	Pop the dividend and divisor, divide, and push the remainder
5	GOTO Lab	Unconditionally jump to the instruction at address Lab
6	BEQZ Lab	Pop the top location and jump to Label if the popped location is zero
7	BNEZ Lab	Pop the top location and jump to Lab if the popped location is not zero
8	BGEZ Lab	Pop the top location and jump to Lab if the popped location is greater than or equal to 0
9	BLTZ Lab	Pop the top location and jump to Lab if the popped location is less than 0

Instruction Set (2)

Stack-based Machine

op	Instruction	Explanation
10	JNS Lab	Push the return address and transfer the control to the instruction at address Lab
11	JR nLoc	Pop the return address into PC and decrement SP by nLoc
12	PUSH FP	Push the content of FP on stack
13	PUSH FP+imm	Push M[FP+imm]
14	PUSH imm	Push a 16-bit integer value
15	PUSH Var	PUSH M[Var]
16	PUSHI Var	Push M[M[Var]]
17	POP FP	FP ← POP()
18	POP FP+imm	M[FP+imm] ← POP()
19	POP Var	M[Var] ← POP()
20	POPI Var	M[M[Var]] ← POP()
21	SWAP	Swaps two top words on the stack
22	MOVE	FP ← SP
23	ISP nLoc	Increase/decrease SP by nLoc
24	READ	Read an input and push it on stack
25	PRNT	Pop the top location and print it
26	STOP	Terminate the program

Instruction Set

Accumulator-based

Imm:	16-bit 2's compliment
AC:	Accumulator
PC:	Program counter
Var:	Variable
Lab:	Label
M[A]:	Memory content of variable A
PUSH PC:	Push PC on stack
POP:	Remove top content on stack
SP:	Reserved location, Stack pointer
ZERO:	Reserved location, M[ZERO]=0

$+Imm: Local variable
Its address is M[SP]+Imm,
where Imm is a 16-bit integer.

Example: ADD $+4 means
AC ← AC + M[M[SP]+4]

op	Instruction	Explanation
0	LI Imm	AC ← Imm
1	ADDI Imm	AC ← AC+Imm
2	ADD Var	AC ← AC+M[Var]
3	SUB Var	AC ← AC-M[Var]
4	MUL Var	AC ← AC*M[Var]
5	DIV Var	AC ← AC/M[Var]
6	REM Var	AC ← AC%M[Var]
7	GET Var	AC ← M[Var]
8	PUT Var	M[Var] ← AC
9	GOTO Lab	PC ← Lab
10	BEQZ Lab	If AC = 0 GOTO Lab
11	BNEZ Lab	If AC ≠ 0 GOTO Lab
12	BGEZ Lab	If AC ≥ 0 GOTO Lab
13	BLTZ Lab	If AC < 0 GOTO Lab
14	JNS Lab	PUSH PC & PC ← Lab
15	JR	PC ← M[M[SP]] & POP
16	READ	AC ← Input
17	PRNT	Print AC
18	STOP	Terminate program
19	GETI Var	AC ←M[M[Var]]
20	PUTI Var	M[M[Var]] ← AC

Pseudo-Instructions

Accumulator-based

	Pseudo-instruction	Meaning	Instruction
1	POP	M[SP] ← M[SP] - 1	GET SP ADDI -1 PUT SP
2	TOP Var	M[Var] ← M[M[SP]]	GETI SP PUT Var
3	PUSH Var	M[SP] ← M[SP] + 1 M[M[SP]] ← M[Var]	GET SP ADDI 1 PUT SP GET Var PUTI SP
4	LA Var	AC ← Address of Var	LI Var

Instruction Set

Two-Address M2M

Imm:	16-bit 2's compliment
PC:	Program counter
M[A]:	Memory content of variable A
SP:	Reserved location, Stack pointer
ZERO:	Reserved location, M[ZERO]=0
INPUT:	Reserved location for input
OUTPUT:	Reserved location for output

$+Imm:	Local variable Its address is M[SP]+Imm, where Imm is a 16-bit integer.
Example:	ADD Var $+4 means M[Var] = M[Var] + M[M[SP]+4]

op	Instruction	Meaning
0	LI C Imm	M[C] ← Imm
1	ADDI C Imm	M[C] ← M[C]+Imm
2	ADD C A	M[C] ← M[C]+M[A]
3	SUB C A	M[C] ← M[C]-M[A]
4	MUL C A	M[C] ← M[C]*M[B]
5	DIV C A	M[C] ← M[C]/M[B]
6	REM C A	M[C] ← M[C]%M[B]
7	GET C A	M[C] ← M[M[A]]
8	PUT B A	M[M[B]] ← M[A]
9	GOTO L	PC ← L
10	BEQZ A L	If M[A] = 0 GOTO L
11	BNEZ A L	If M[A] ≠ 0 GOTO L
12	BGEZ A L	If M[A] ≥ 0 GOTO L
13	BLTZ A L	If M[A] < 0 GOTO L
14	JNS L	M[*SP*] = M[*SP*]+1, M[M[*SP*]] = *PC*, & *PC* ← L
15	JR	*PC* ← M[M[*SP*]] & M[*SP*] = M[*SP*]-1
16	READ	M[*INPUT*] ← input
17	PRNT	Print M[*OUTPUT*]
18	STOP	Stop

Pseudo-Instructions

Two-Address M2M

	Pseudo-instruction	Meaning	Instruction
1	MOVE A B	M[A] ← M[B]	ADD ZERO B LI A 0 ADD A ZERO LI ZERO 0
2	NEG A	M[A] ←-M[A]	SUB ZERO A LI A 0 ADD A ZERO LI ZERO 0
3	POP A	M[A] ← M[M[SP]] M[SP] ← M[SP] – 1	GET A SP ADDI SP -1
4	PUSH A	M[SP] ← M[SP] + 1 M[M[SP]] ← M[A]	ADDI SP 1 PUT A SP
5	LA A B	M[A] ← address of B	LI A B

Instruction Set

Three-Address M2M

Imm:	32-bit 2's compliment
PC:	Program counter
M[A]:	Memory content of variable A
SP:	Reserved location, Stack pointer
ZERO:	Reserved location, M[ZERO]=0
INPUT:	Reserved location for input
OUTPUT:	Reserved location for output

$+Imm:	Local variable Its address is M[SP]+Imm, where Imm is a 16-bit integer.
Example: ADD A B $+4 means	M[A] = M[B] + M[M[SP]+4]

op	Instruction	Meaning
0	LI C Imm	M[C] ← Imm
1	ADDI C A Imm	M[C] ← M[A]+Imm
2	ADD C A B	M[C] ← M[A]+M[B]
3	SUB C A B	M[C] ← M[A]-M[B]
4	MUL C A B	M[C] ← M[A]*M[B]
5	DIV C A B	M[C] ← M[A]/M[B]
6	REM C A B	M[C] ← M[A]%M[B]
7	GET C A B	M[C] ← M[A+M[B]]
8	PUT C A B	M[A+M[B]] ← M[C]
9	GOTO L	PC ← L
10	BEQ A B L	If M[A] = M[B] GOTO L
11	BNE A B L	If M[A] ≠ M[B] TO L
12	BGE A B L	If M[A] ≥ M[B] GOTO L
13	BLT A B L	If M[A] < M[B] GOTO L
14	JNS L	M[*SP*] = M[*SP*]+1, M[M[*SP*]] = *PC*, & *PC* ← L
15	JR	*PC* ← M[M[*SP*]] & M[*SP*] = M[*SP*]-1
16	READ	M[INPUT] ← Input
17	PRNT	Print M[OUTPUT]
18	STOP	Stop

Pseudo-Instructions

Three-Address M2M

	Pseudo-instruction	Meaning	Instruction
1	MOVE A B	M[A] ← M[B]	ADD A ZERO B
2	GETI A B	M[A] ← M[M[B]]	GET A ZERO B
3	PUTI A B	M[M[B]] ← M[A]	PUT A ZERO B
4	BEQZ A L	If M[A] = 0 GOTO L	BEQ A ZERO L
5	BNEZ A L	If M[A] ≠ 0 GOTO L	BNE A ZERO L
6	BGEZ A L	If M[A] ≥ 0 GOTO L	BGE A ZERO L
7	BLTZ A L	If M[A] < 0 GOTO L	BLT A ZERO L
8	NEG A	M[A] ← - M[A]	SUB A ZERO A
9	POP A	M[A] ← M[M[SP]] M[SP] ← M[SP] – 1	GET A ZERO SP ADDI SP SP -1
11	PUSH A	M[SP] ← M[SP] + 1 M[M[SP]] ← M[A]	ADDI SP SP 1 PUT A ZERO SP
12	LA A B	M[A] ← address of B	LI A B

32 General-Purpose Registers

MIPS Register Convention

Name	Number	Usage
$zero	$0	The constant value 0
$at	$1	Reserved for assembler
$v0-$v1	$2-$3	Expression evaluation and results of a function
$a0-$a3	$4-$7	Argument 1-4
$t0-$t7	$8-$15	Temporary (not preserved across call)
$s0-$s7	$16-$23	Saved temporary (preserved across call)
$t8-$t9	$24-$25	Temporary (not preserved across call)
$k0-$k1	$26-$27	Reserved for OS kernel
$gp	$28	Pointer to global area
$sp	$29	Stack pointer
$fp	$30	Frame pointer
$ra	$31	Return address (used by function call)

Instruction Set

Two-Address R2R

Imm:	16-bit 2's compliment
PC:	Program counter
R, R1:	Registers
L:	Label
M[R]:	Memory content at address R

op	Instruction	Meaning
0	LI R Imm	R ← Imm
1	ADDI R Imm	R ← R+Imm
2	ADD R R1	R ← R+R1
3	SUB R R1	R ← R-/R1
4	MUL R R1	R ← R*R1
5	DIV R R1	R ← R/R1
6	REM R R1	R ← R%R1
7	GET R R1	R ← M[R1]
8	PUT R R1	M[R1] ← R
9	GOTO L	PC ← L
10	BEQZ R L	If R = 0 GOTO L
11	BNEZ R L	If R ≠ 0 GOTO L
12	BGEZ R L	If R ≥ 0 GOTO L
13	BLTZ R L	If R < 0 GOTO L
14	JNS L	$ra ← *PC* & *PC*←L
15	JR	PC ← $ra
16	READ	$v0 ← Input
17	PRNT	Print $a0
18	STOP	Stop

Pseudo-Instructions

Two-Address R2R

	Pseudo-instruction	Meaning	Instruction
1	LA R Var	R ← &Var	LI R Var
2	MOVE R R1	R ← R1	LI $at 0 ADD $at R1 LI R 0 ADD R $at
3	NEG R	R ← -R	LI $at 0 SUB $at R LI R 0 ADD R $at
4	GETI R2 Imm	R2 ← M[Imm]	LI $at Imm GET R2 $at
5	PUTI R2 Imm	M[Imm] ← R2	LI $at Imm PUT R2 $at
6	GETV R2 Var	R2 ← M[Var]	LI $at var GET R2 $at
7	PUTV R2 Var	M[Var] ← R2	LI $at var PUT R2 $at
8	POP R	R ← M[$sp] $sp = $sp - 1	GET R $sp ADDI $sp -1
9	PUSH R	$sp = $sp + 1 M[$sp] ← R	ADDI $sp 1 PUT R $sp

Instruction Set

Two-Address M2R

Imm:	16-bit 2's compliment
PC:	Program counter
R, R1:	Registers
L:	Label
A:	Variable
M[R]:	Memory content at address R
M[A]:	Memory content at address A

op	Instruction	Meaning
0	LI R Imm	R ← Imm
1	ADDI R Imm	R ← R+Imm
2	ADD R A/R1	R ← R+M[A]/R1
3	SUB R A/R1	R ← R-M[A]/R1
4	MUL R A/R1	R ← R*M[A]/R1
5	DIV R A/R1	R ← R/M[A]/R1
6	REM R A/R1	R ← R%M[A]/R1
7	GET R A/R1	R ← M[A/R1]
8	PUT R A/R1	M[A/R1] ← R
9	GOTO L	PC ← L
10	BEQZ R L	If R = 0 GOTO L
11	BNEZ R L	If R ≠ 0 GOTO L
12	BGEZ R L	If R ≥ 0 GOTO L
13	BLTZ R L	If R < 0 GOTO L
14	JNS L	$ra ← *PC* & *PC*←L
15	JR	PC ← $ra
16	READ	$v0 ← Input
17	PRNT	Print $a0
18	STOP	Stop

Pseudo-Instructions

Two-Address M2R

	Pseudo-instruction	Meaning	Instruction
1	LA R Var	R ← &Var	LI R Var
2	MOVE R R1	R ← R1	LI $at 0 ADD $at R1 LI R 0 ADD R $at
3	NEG R	R ← -R	LI $at 0 SUB $at R LI R 0 ADD R $at
4	GETI R2 Imm	R2 ← M[Imm]	LI $at Imm GET R2 $at
5	PUTI R2 Imm	M[Imm] ← R2	LI $at Imm PUT R2 $at
6	GETV R2 Var	R2 ← M[Var]	LI $at var GET R2 $at
7	PUTV R2 Var	M[Var] ← R2	LI $at var PUT R2 $at
8	POP R	R ← M[$sp] $sp = $sp - 1	GET R $sp ADDI $sp -1
9	PUSH R	$sp = $sp + 1 M[$sp] ← R	ADDI $sp 1 PUT R $sp

Instruction Set

Three-Address R2R

Imm:	16-bit 2's compliment
offset:	Imm or address of variable
PC:	Program counter
R, R1:	Registers
L:	Label
M[R]:	Memory content at address R

op	Instruction	Meaning
0	LI R Imm	R ← Imm
1	ADDI R R1 Imm	R ← R1+Imm
2	ADD R R1 R2	R ← R1 + R2
3	SUB R R1 R2	R ← R1 + R2
4	MUL R R1 R2	R ← R1 + R2
5	DIV R R1 R2	R ← R1 + R2
6	REM R R1 R2	R ← R1 + R2
7	GET R R1 offset	R ← M[R1+ offset]
8	PUT R R1 offset	M[R1+ offset] ← R
9	GOTO L	PC ← L
10	BEQ R R1 L	If R = R1 GOTO L
11	BNE R R1 L	If R ≠ R1 GOTO L
12	BGE R R1 L	If R ≥ R1 GOTO L
13	BLT R R1 L	If R < R1 GOTO L
14	JNS L	$ra ← *PC* & *PC* ←L
15	JR	PC ← $ra
16	READ	$v0 ← Input
17	PRNT	Print $a0
18	STOP	Terminate

Pseudo-Instructions

Three-Address R2R

	Pseudo-instruction	Meaning	Instruction
0	LA R Var	R ← &Var	Var: variable, 16-bit address
1	MOVE R2 R1	R2 ← R1	ADD R2 R1 $zero
2	NEG R	R ← - R	SUB R $zero R
3	GETR R2 R1	R2 ← M[R1]	GET R2 R1 0
4	PUTR R2 R1	M[R1] ← R2	PUT R2 R1 0
5	GETI R2 imm	R2 ← M[imm]	GET R2 $zero imm
6	PUTI R2 imm	M[imm] ← R2	PUT R2 $zero imm
7	GETV R2 Var	R2 ← M[Var]	GET R2 $zero var
8	PUTV R2 Var	M[Var] ← R2	PUT R2 $zero var
9	BEQZ R L	If R = 0 GOTO L	BEQ R $zero L
10	BNEZ R L	If R ≠ 0 GOTO L	BNE R $zero L
11	BGEZ R L	If R ≥ 0 GOTO L	BGE R $zero L
12	BLTZ R L	If R < 0 GOTO L	BLT R $zero L
13	POP R	R ← M[SP] SP ← SP – 1	GET R $sp 0 ADDI $sp $sp -1
14	PUSH R	SP ← SP + 1 M[SP] ← R	ADDI $sp %sp 1 PUT R $sp 0