Computer Organization and Architecture

Lecture – 14

Oct 5th , 2022

LEGv8 Assembly Language

1. Arithmetic and Immediate instructions
2. Load/Store Instructions
    1. Accessing operands in memory
3. Decision Making
    1. Branch on zero/not zero (==, ! =)
    2. Condition flags, branches (<, >, >=, ==, etc.)
4. Procedures
    1. Branch and link (BL), Branch register (BR)
    2. Spilling registers

Procedures

```
2  int main () {
3
4     int a = 100;
5     int b = 200;
6     int ret;
7
8     ret = add(a, b);
9
10    return 0;
11 }
12
13
14 int add(int num1, int num2) {
15
16    int result;
17
18    result = num1 + num2
19
20    return result;
21 }
```

Procedure Calling

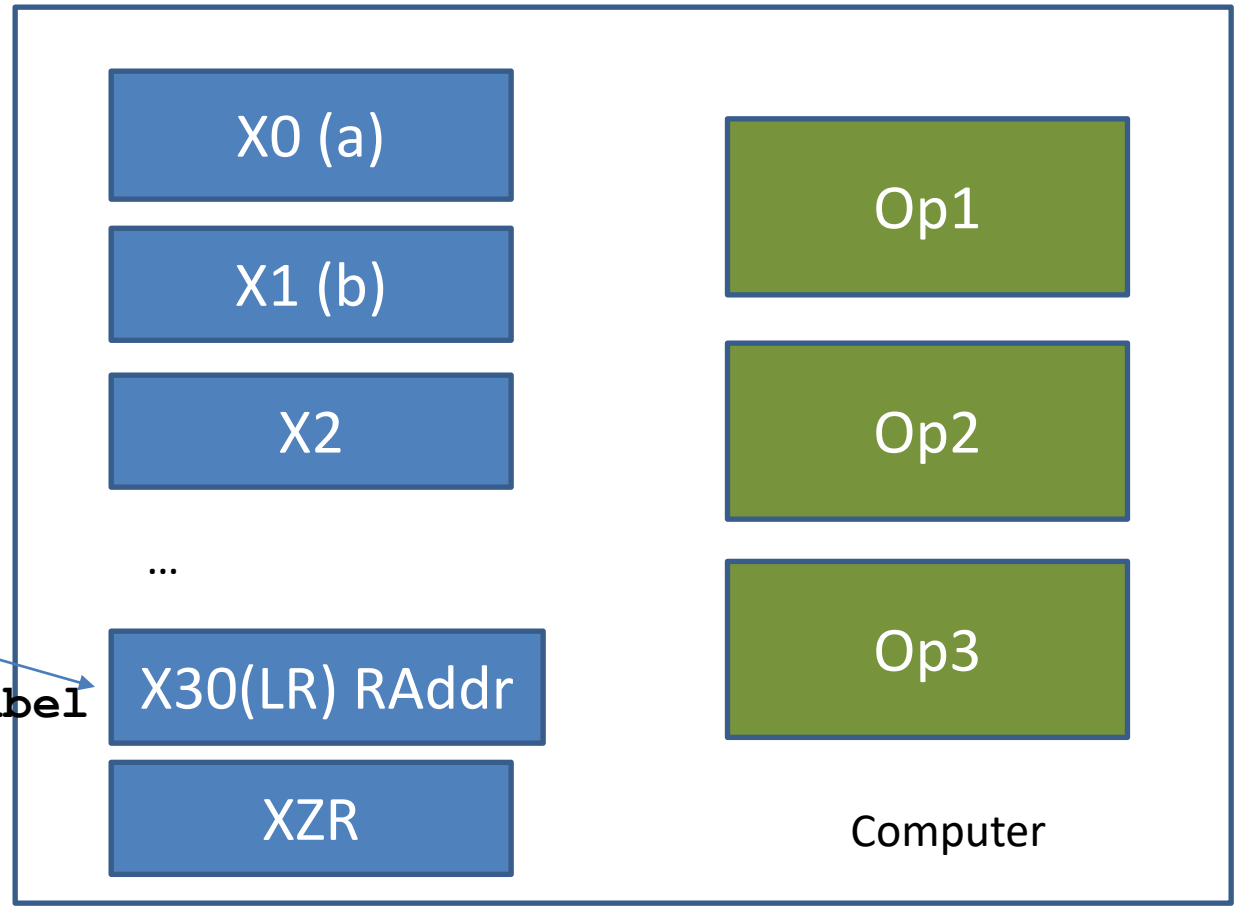
- Steps required
  1. Place parameters in registers X0 to X7
  2. Transfer control to procedure
  3. Acquire storage for procedure
  4. Perform procedure's operations
  5. Place result in register for caller
  6. Return to place of call (address in X30)

Procedure Calling

- Steps required
  1. Place parameters in registers X0 to X7
  2. **Transfer control to procedure (BL, branch and link)**
  3. Acquire storage for procedure
  4. Perform procedure's operations
  5. Place result in register for caller
  6. Return to place of call (address in X30)

Steps in Executing a Procedure

```
2  int main () {
3
4      int a = 100;
5      int b = 200;
6      int ret;
7  ---------------------------------
8      ret = add(a, b);
9
10     return 0;
11 }
12
13
14 int add(int num1, int num2) {
15
16     int result;
17
18     result = num1 + num2
19
20     return result;
21 }
```



BL ProcedureLabel

1. Caller puts parameters in registers for the procedure
    1. Specify a return address

Procedure Instructions

- Procedure call: jump and link

  `BL ProcedureLabel`

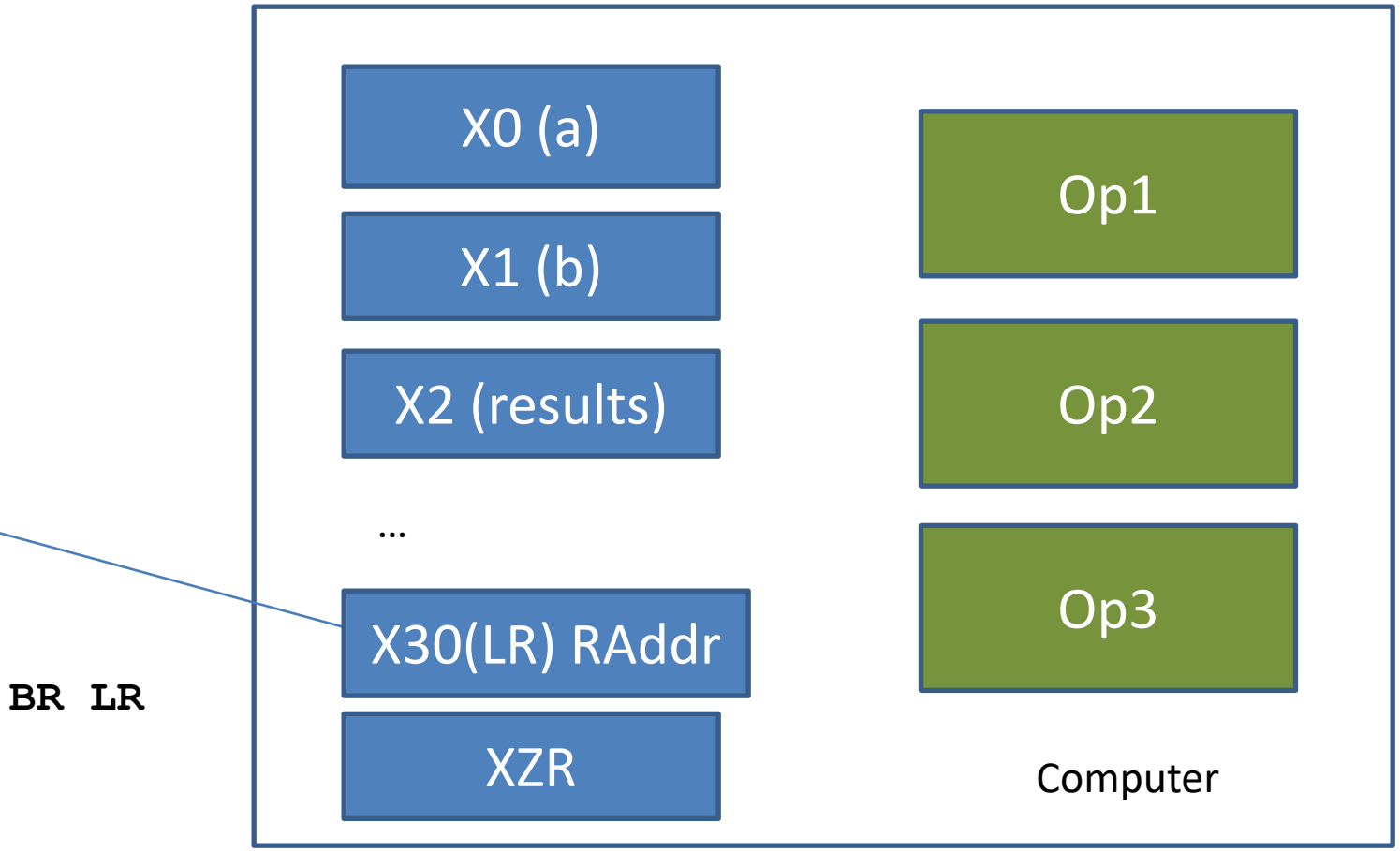
  – Address of following instruction put in X30 (LR)
    - Actually PC + 4 (32 bit instruction)
  – Jumps to target address

Procedure Calling

- Steps required
  1. Place parameters in registers X0 to X7
  2. Transfer control to procedure (BL, branch and link)
  3. Acquire storage for procedure
  4. Perform procedure's operations
  5. Place result in register for caller
  6. **Return to place of call (BR, branch register)**

Steps in Executing a Procedure

```
2  int main () {
3
4     int a = 100;
5     int b = 200;
6     int ret;
7
8     ret = add(a, b);
9
10    return 0;
11 }
12
13
14 int add(int num1, int num2) {
15
16    int result;
17
18    result = num1 + num2
19
20    return result;
21 }
```



BR LR

6. Return control to Caller

Procedure Instructions

- Procedure call: jump and link

  `BL ProcedureLabel`

  – BL: Branch and Link Register
  – Address of following instruction put in X30 (LR)
  – Jumps to target address
- Procedure return: jump register

  `BR LR`

  – BR: Branch Register
  – Copies LR to program counter

Example

```
int main(){
    result = add(a, b)
}

Procedure:
Int add(int c, int d){
        c = c + d
        return c
}
```

Assume that the variable a, b and result correspond to registers X9, X10, X5. What is the LEGv8 code

What Registers used?

- X0 – X7: procedure arguments/results
- X30 (LR): link register (return address)
  - Also called as program counter (PC)

Example

```
int main(){
    result = add(a, b)
}

Procedure:
Int add(int c, int d){
        c = c + d
        return c
}
```

Assume that the variable a, b and result correspond to registers X9, X10, X5. What is the LEGv8 code

ADD X0, X9, XZR //Move a to X0
ADD X1, X10, XZR // Move b to x1
BL AddProcedure //Call the procedure and link the return address

AddProcedure: ADD X0, X0, X1 // c = c+d
ADD X5, X0, XZR // Copy c into result register
BR LR // Branch to caller address (PC +4)

Spill and Restore Registers

The caller/callee have to save some registers before execution

1. Save variable to memory from register
2. Finish executing procedure
3. Restore value of variable from memory to Previous location

Values are store in a *Stack*

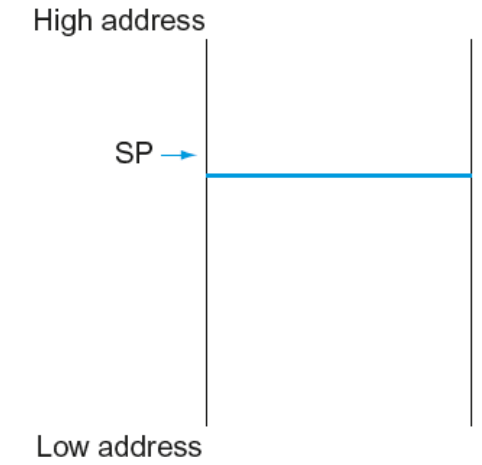
Stack Pointer (X28): Register containing memory location to store the values

Spilling to stack

- To spill registers (X10, X9, X19) on to the stack

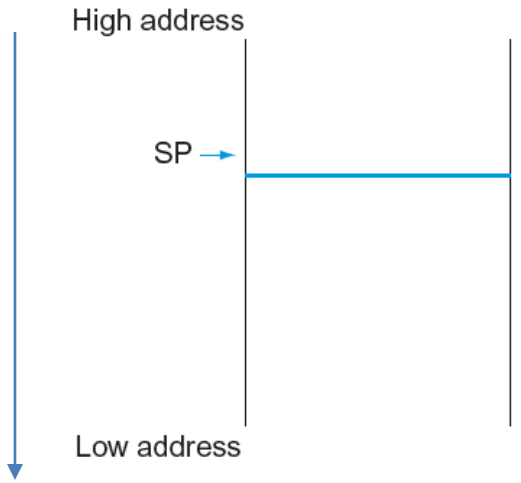
Spilling to stack

- To spill registers (X10, X9, X19) on to the stack
- Address of stack is save in X28 (SP)



Spilling to stack

- To spill registers (X10, X9, X19) on to the stack
- Address of stack is save in X28 (SP)
- Goes from High to low for historic reasons

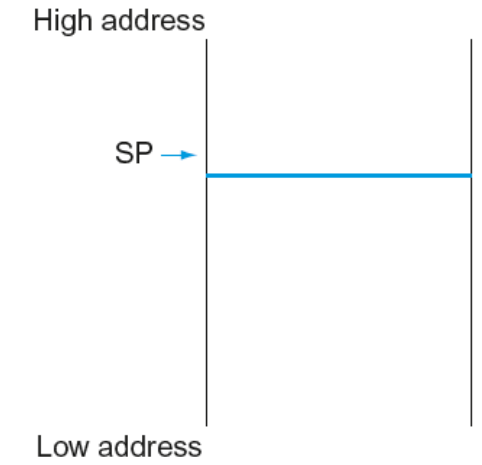


Spilling to stack

- To spill registers (X10, X9, X19) on to the stack
- Address of stack is save in X28 (SP)

LEGv8 Code:

Make room for three items

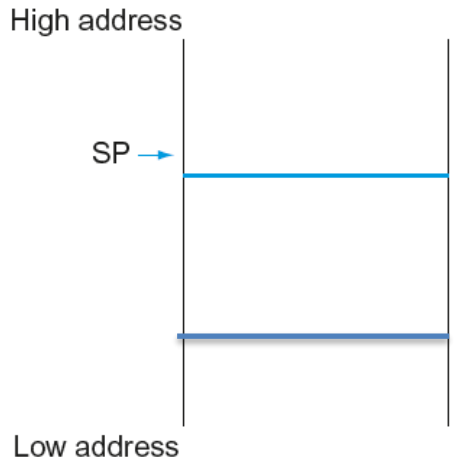


Spilling to stack

- To spill registers (X10, X9, X19) on to the stack
- Address of stack is save in X28 (SP)

LEGv8 Code:

$$// \ Make \ room \ for \ three \ items$$
$$SUBI \ SP, SP, \#24$$



Spilling to stack

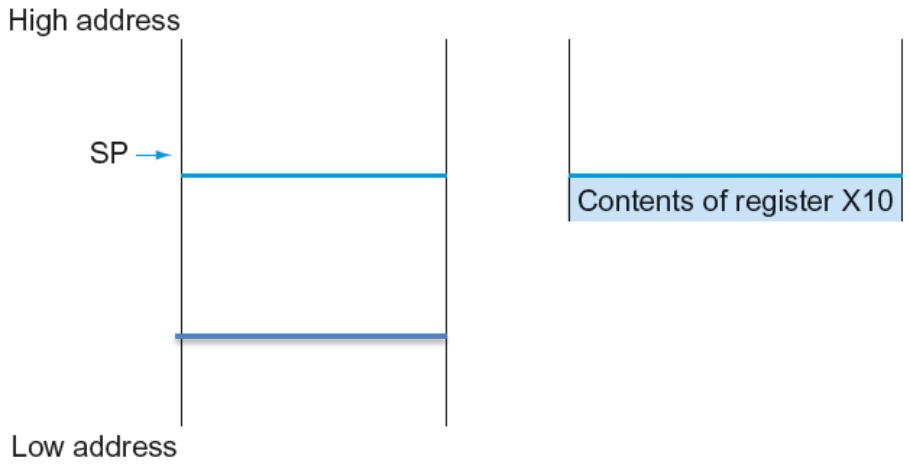
- To spill registers (X10, X9, X19) on to the stack
- Address of stack is save in X28 (SP)

LEGv8 Code:

$SUBI\ SP, SP, \#24\ //\ Make\ room\ for\ three\ items$

$STUR\ X10, [SP, \#16]//Spill\ X10$

(PUSH)

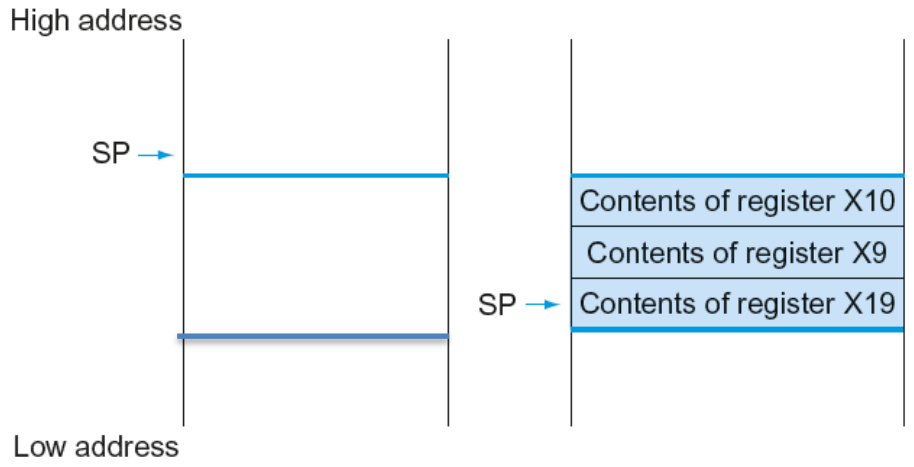


Spilling to stack

- To spill registers (X10, X9, X19) on to the stack
- Address of stack is save in X28 (SP)

LEGv8 Code:

$SUBI\ SP, SP, \#24\ //\ Make\ room\ for\ three\ items$
$STUR\ X10, [SP, \#16]//Spill\ X10$
$STUR\ X9, [SP, \#8]//Spill\ X9$
$STUR\ X19, [SP, \#0]//Spill\ X19$



Restore from Stack

- To spill registers (X10, X9, X19) on to the stack
- Address of stack is save in X28 (SP)

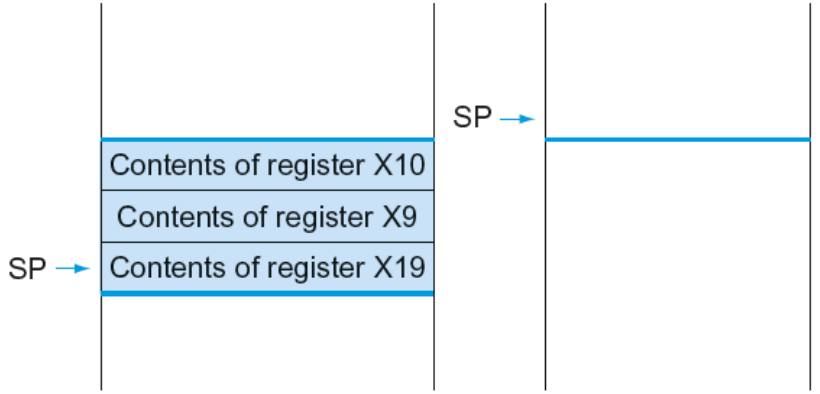
LEGv8 Code:

$LDUR\ X19, [SP, \#0] //Spill\ X\ 19$ ***(POP)***

$LDUR\ X9, [SP, \#8] //Spill\ X\ 9$

$LDUR\ X10, [SP, \#16] //Spill\ X10$

$ADDI\ SP, SP, \#24\ //\ Make\ room\ for\ three\ items$



Example

```
int main(){
    result = add(a, b)
..
}

Procedure:
Int add(int c, int d){
        c = c + d
        return c
}
```

Assume that the variable a, b and result correspond to registers X9, X10, X5.
Assume that the main needs to spill the registers containing the values **a** and **b** before making a call to the add function.
And finally restore them once the add procedure is executed.
What is the LEGv8 code?

```
int main(){
    result = add(a, b)
}

Procedure:
Int add(int c, int d){
        c = c + d
        return c
}
```

Assume that the variable a, b and result correspond to registers X9, X10, X5.
Assume that the main procedure needs to spill the registers containing the values **a** and **b** before making a call to the add function.
And finally restore them once the add procedure is executed.
What is the LEGv8 code?

```
ADD X0, X9, XZR //Move a to X0
ADD X1, X10, XZR // Move b to x1
SUBI SP, SP, #16 // Make room for two items
STUR X9, [SP, #8]//Spill X10
STUR X10, [SP, #0]//Spill X9
BL AddProcedure //Call the procedure and link the return address

AddProcedure: ADD X0, X0, X1 // c = c+d
ADD X5, X0, XZR // Copy c into result register
BR XLR // Branch to caller address (PC +4 )

LDUR X10, [SP, #0]//Restore X10
LDUR X9, [SP, #8]//Restore X9
ADDI SP, SP, #16 // Reset SP
```

Registers to be Saved

- X9 to X17: temporary registers
  - The caller has to save them if needed for latter
  - Not preserved by the callee
- X19 to X28: saved registers
  - If used, the callee saves and restores them

Example

```
int main(){
    result = add(a, b)
..
}

Procedure:
Int add(int c, int d){
        c = c + d
        return c
}
```

Assume that the variable a, b and result correspond to registers X9, X10, X5.
Assume that the main needs to spill the registers containing the values **a** and **b** before making a call to the add function, and finally restore them once the add procedure is executed.
In addition assume that the caller needs to use the saved register X19
What is the LEGv8 code?

```
int main(){
    result = add(a, b)
}

Procedure:
Int add(int c, int d){
        c = c + d
        return c
}
```

ADD X0, X9, XZR //Move a to X0
ADD X1, X10, XZR // Move b to x1
$SUBI\ SP, SP, \#16\ //\ Make\ room\ for\ two\ items$
$STUR\ X9, [SP, \#8]//Spill\ X10$
$STUR\ X10, [SP, \#0]//Spill\ X9$
BL AddProcedure //Call the procedure and link the return address

AddProcedure: ADD X0, X0, X1 // c = c+d
$\boldsymbol{SUBI\ SP, SP, \#8\ //\ Make\ room\ for\ one\ item}$
$\boldsymbol{STUR\ X19\ [SP, \#0]//Spill\ X19}$
ADD X5, X0, XZR // Copy c into result register
$\boldsymbol{LDUR\ X19, [SP, \#0]//Restore\ X19}$
$\boldsymbol{ADDI\ SP, SP, \#8\ //\ Reset\ SP}$
BR XLR // Branch to caller address (PC +4)

$LDUR\ X10, [SP, \#0]//Restore\ X10$
$LDUR\ X9, [SP, \#8]//Restore\ X9$
$ADDI\ SP, SP, \#16\ //\ Reset\ SP$