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

Instructions for Making Decisions

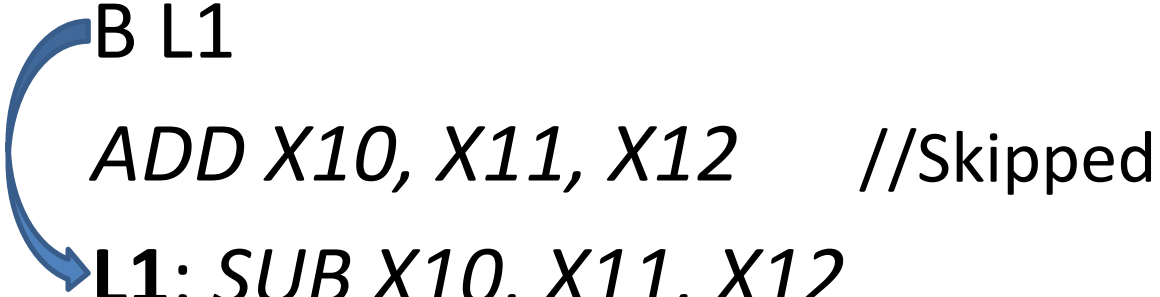
- Define Labels for instructions.
- LEGv8 Code:

  L1: *ADD X9, X21, X9*

- Unconditional Branch: Instruct computer to branch to label
- B – branch to label
- LEGv8 Code:

  B L1 `// Branch to statement with label L1`

Example



Instructions for Making Decisions

- Define Labels for instructions.
- LEGv8 Code:

  L1: *ADD X9, X21, X9*

- Instruct computer to branch to instruction using the label if some condition is satisfied.
- CBZ – compare and branch if zero
- CBNZ – compare and branch if not zero
- LEGv8 Code:

  $CBZ\ \ register,\ \ L1$ // if (register == 0) branch to instruction labeled L1;

  $CBNZ\ \ register,\ \ L1$ // if (register != 0) branch to instruction labeled L1;

Example

CBZ X9, L2

Checks to see if the value in register is 0
If yes, branches to L2

L1: *ADD X10, X11, X12*

L2: *SUB X10, X11, X12*

Example

CBZ X9, L2

L1: *ADD X10, X11, X12* Skipped

L2: *SUB X10, X11, X12*

Register X9
0

Example

CBZ X9, L2

L1: *ADD X10, X11, X12*

L2: *SUB X10, X11, X12*

Register X9
1

Example

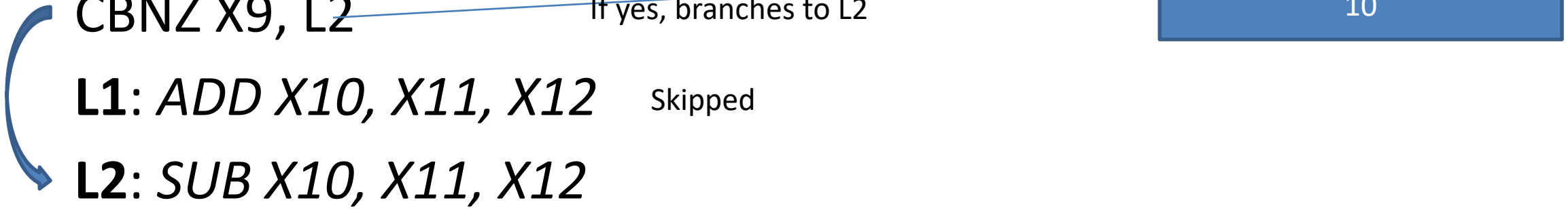
CBNZ X9, L2

L1: *ADD X10, X11, X12*

L2: *SUB X10, X11, X12*

Checks to see if the
value in register is not 0
If yes, branches to L2

Example


CBNZ X9, L2
L1: ADD X10, X11, X12
L2: SUB X10, X11, X12
Checks to see if the
value in register is not 0
If yes, branches to L2
Register X9
10
Skipped

Example

Let the variables x and f correspond to registers X9 and X10

```
If (x == 0)
        f = f + 1
else
        f = f - 1
```

Example

Let the variables x and f correspond to registers X9 and X10

```
If (x == 0)
        f = f + 1
else
        f = f - 1
```

```
CBNZ X9, L1
ADDI X10, X10, #1
B Exit
L1: SUBI X10, X10, #1
Exit:
```

Example

Let the variables x and f correspond to registers X9 and X10

```
If (x == 0)
        f = f + 20
else if (x == 1)
        f = f – 1
else
        f = f + 1
```

Example

Let the variables x and f correspond to registers X9 and X10

```
If (x == 0)
        f = f + 1
else if (x == 1)
        f = f – 1
else
        f = f + 20
```

```
        CBNZ X9, L1
        ADDI X10, X10, #1
        B EXIT

L1:     SUBI X11, X9, 1
        CBNZ X11, L2
        SUBI X10, X10, #1
        B EXIT

L2:     ADDI X10, X10, #20
Exit:
```

Example

Let the variables x and f correspond to registers X9 and X10

```
If (x == 0)
        f = f + 1
else if (x == 1)
        f = f – 1
else
        f = f + 20
```

```
        CBNZ X9, L1
        ADDI X10, X10, #1
        B EXIT

L1:     SUBI X11, X9, 1
        CBNZ X11, L2
        SUBI X10, X10, #1
        B EXIT

L2:     ADDI X10, X10, #20
Exit:
```

Example

Let the variables x and f correspond to registers X9 and X10

```
If (x == 0)
        f = f + 1
else if (x == 1)
        f = f – 1
else
        f = f + 20
```

```
        CBNZ X9, L1
        ADDI X10, X10, #1
        B EXIT

L1:     SUBI X11, X9, 1
        CBNZ X11, L2
        SUBI X10, X10, #1
        B EXIT

L2:     ADDI X10, X10, #20
Exit:
```

Compiling Loop Statements

- C code:

```
while (True)
        k = k + 1
```

k in x24

- Compiled LEGv8 code:

Compiling Loop Statements

- C code:

```
while (True)
        k = k + 1
```

k in x24

- Compiled LEGv8 code:

```
Loop: ADDI X24, X24, #1
      B     Loop
```

Example

- C code:

```
while (True)
      k = k + 1
    if (k == 10)
          break
```

k in x24

Example

- C code:

```
while (True)
      k = k + 1
    if (k == 10)
        break
```

k in x24

```
Loop:  ADDI X24, X24, #1
       SUBI X25, X24, #10
       CBZ X25, Exit
       B    Loop
Exit:
```

Compiling Loop Statements

- C code:

```
while (save[i] == k) i += 1;
```

  - i in x22, k in x24, address of save in x25

- Compiled LEGv8 code:

```
?
```

Compiling Loop Statements

- C code:

```
while (save[i] == k) i += 1;
```

  - i in x22, k in x24, address of save in x25

- Compiled LEGv8 code:

```
Loop: LSL    X10,X22,#3      // X10 = i*2^3
      ADD    X10,X10,X25     // Address to load save[i]
      LDUR   X9,[X10,#0]     // load save[i]
      SUB    X11,X9,X24      // X11 = save[i] - k
      CBNZ   X11,Exit        // conditional branch
      ADDI   X22,X22,#1      // i += 1
      B      Loop            // uncond. branch
Exit: …
```

Using Condition Codes/Flags for Comparisons

$<$	Less than
$\leq$	Less than or equal
$>$	Greater than
$\geq$	Greater than or equal
$=$	Equal
$!=$	Not equal

Condition code

- LEGv8 provides four added bits called condition codes.
- Some arithmetic instructions can optionally set these flags based on the result of the operation.
- Then the branch (B) instruction can check these bits to do comparisons.

Condition codes/flags

$Negative$(N)	
$Zero$ (Z)	
$Overflow$ (V)	
$Carry$ (C)	

Set Flag Instructions

Arithmetic Instruction	With Set Flag Option (Suffix S)	Description
ADD	**ADDS**	**Add and set condition flag**
ADDI	**ADDIS**	**Add immediate and set condition flag**
SUB	**SUBS**	**Subtract and set condition flag**
SUBI	**SUBIS**	**Subtract immediate and set condition flag**
AND	**ANDS**	**AND and set condition flag**
ANDI	**ANDIS**	**AND immediate and set condition flag**

Example SUBS : Subtract and Set Flag

- LEGv8 provides set flag variants for SUB

$Assume\ i = +9$, j = +10 are signed integers, and store in X1, and X2 respectively

To do the comparison

$$If\ (i < j)$$

…

Condition codes/flags

$Negative$(N)	
$Zero$ (Z)	
$Overflow$ (V)	
$Carry$ (C)	

Example SUBS : Subtract and Set Flag

- LEGv8 provides set flag variants for SUB

$Assume\ i = +9$, j = +10 are signed integers, and store in X1, and X2 respectively

To do the comparison

$$If\ (i < j)$$

…

LEGv8 code:

$$SUBS\ X1, X1, X2$$

Condition codes/flags

$Negative$(N)	
$Zero$ (Z)	
$Overflow$ (V)	
$Carry$ (C)	

Example SUBS : Subtract and Set Flag

- LEGv8 provides set flag variants for SUB

$Assume\ i = +9$, j = +10 are signed integers, and store in X1, and X2 respectively

To do the comparison

$$If\ (i < j)$$

…

LEGv8 code:

$$SUBS\ X1, X1, X2$$

Result is -1
The N flag is set

Condition codes/flags

$Negative$(N)	1
$Zero$ (Z)	
$Overflow$ (V)	
$Carry$ (C)	

Example SUBS : Subtract and Set Flag

- LEGv8 provides set flag variants for SUB

$Assume\ i = +9$, j = +10 are signed integers, and store in X1, and X2 respectively

To do the comparison

$$If\ (i < j)$$

…

LEGv8 code:

$$SUBS\ X1, X1, X2$$
$$//\ Branch\ if\ N\ flag\ is\ set$$

Condition codes/flags

$Negative$(N)	1
$Zero$ (Z)	
$Overflow$ (V)	
$Carry$ (C)	

Conditional branches use these codes to do comparisons

Conditional Branches that use Flags

- Format ➔ B.cond
- Use subtract to set flags and then conditionally branch
  - **B.EQ**
  - **B.NE**
  - **B.LT** (less than, **signed**)
  - **B.LO** (less than, unsigned)
  - **B.LE** (less than or equal, **signed**)
  - **B.LS** (less than or equal, unsigned)
  - **B.GT** (greater than, **signed**)
  - **B.HI** (greater than, unsigned)
  - **B.GE** (greater than or equal, **signed**),
  - **B.HS** (greater than or equal, unsigned)

Conditional Example

```
if (a > b)
        a += 1;
```

– a in X22, b in X23

LEGv8 Code:

?

Conditional Example

```
if (a > b)
        a += 1;
```

– a in X22, b in X23

LEGv8 Code:

```
    SUBS X9,X22,X23  // use subtract to make comparison
    B.LTE Exit       // conditional branch
    ADDI X22,X22,#1
Exit:
```

Example

- C code:

```
while (True)
      k = k + 1
    if (k > 10)
        break
```

k in x24

Example

- C code:

```
while (True)
      k = k + 1
    if (k > 10)
          break
```

```
Loop:  ADDI X24, X24, #1
       SUBIS X25, X24, #10
       B.GT Exit
       B     Loop
Exit:
```

k in x24, and is a signed number

Example

- C code:

```
while (True)
      k = k + 1
    if (k > 10)
         break
```

k in x24, and is a signed number

```
Loop:  ADDI X24, X24, #1
       SUBIS X25, X24, #10
       B.GT Exit
       B     Loop
Exit:
```

The result of the subtract instruction is redundant,
B.GT uses the condition flags for branching
For efficiency, we can give the destination register as XZR instead of X25

Example

- C code:

```
while (True)
      k = k + 1
    if (k > 10)
          break
```

```
Loop:  ADDI X24, X24, #1
       SUBIS XZR, X24, #10
       B.GT Exit
       B     Loop
Exit:
```

k in x24, and is a signed number