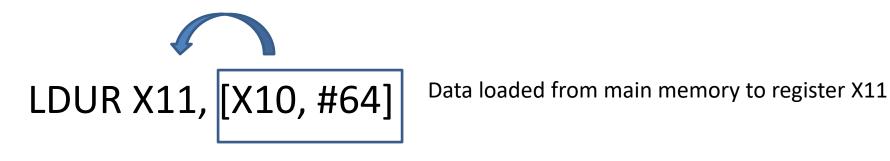
Let the base address for the variable a correspond to register X10.

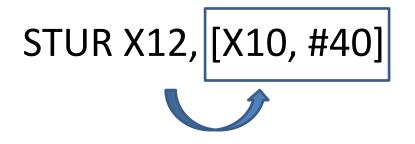
- 1. Load the value a[8] into register X11
- 2. Load the value a[3] into register X11
- 3. Store the value in register X12 in a[7]
- 4. Store the value in register X12 in a[5]

Let the base address for the variable a correspond to register X10.

- 1. Load the value a[8] into register X11 → LDUR X11, [X10, #64]
- 2. Load the value a[3] into register X11 → LDUR X11, [X10, #24]
- 3. Store the value in register X12 in a[7] \rightarrow STUR X12, [X10, #56]
- 4. Store the value in register X12 in a[5] → STUR X12, [X10, #40]

Let the base address for the variable a correspond to register X10.





Data in register X12 is store in main memory

Let the base address for the variable a correspond to register X10.

- 1. Load the value a[8] into register X11 → LDUR X11, [X10, #64]
- 2. Load the value a[3] into register X11 → LDUR X11, [X10, #24]
- 3. Store the value in register X12 in a[7] → STUR X12, [X10, #56]
- 4. Store the value in register X12 in a[5] → STUR X12, [X10, #40]

$$result = a[i]$$

$$result = a[i]$$

- 1. compute i * 8
- 2. Add result to base address
- 3. Load from the resultant address

Logical Shift Left (LSL)

*LSL X*11, *X*19, #*4*// *shift* 4 *bits to left*

Logical Shift Left (LSL)

*LSL X*11, *X*19, #*4*// *shift* 4 *bits to left*

 $144_{ten} = 9_{ten} * 2^4$ Left Shift by *i* bits multiplies by 2^i

• Let the base address for the variable *a, i, results* correspond to register X9, X10, and X11. What is the LEGv8 code for

```
result = a[i]
```

- 1. compute i * 8 (left shift i by 3 bits)
- 2. Add result to base address
- 3. Load from the resultant address

LSL X12, X10, #3 // i*8

$$result = a[i]$$

- 1. compute i * 8 (left shift i by 3 bits)
- 2. Add result to base address
- Load from the resultant address

```
LSL X12, X10, #3 // i*8

ADD X13, X9, X12 // base address (a) + i*8
```

```
result = a[i]
```

- 1. compute i * 8 (left shift i by 3 bits)
- 2. Add result to base address
- 3. Load from the resultant address

```
result = a[i]

i = i + 1

result = result + a[i]
```

```
result = a[i]
i = i + 1
result = result + a[i]
LSL X12, X10, #3 // i*8
ADD X13, X9, X12 // base address (a) + i*8
LDUR X11, [X13, #0] // result = a[i]
```

```
result = a[i]
i = i + 1
result = result + a[i]
LSL X12, X10, #3 // i*8
ADD X13, X9, X12 // base address (a) + i*8
LDUR X11, [X13, #0] // result = a[i]
ADDI X10, X10, #1 // i = i+1
```

```
result = a[i] \\ i = i + 1 \\ result = result + a[i] \\ ADDI X13, X9, X12 // base address (a) + i*8 \\ LDUR X11, [X13, #0] // result = a[i] \\ ADDI X10, X10, #1 // i = i+1 \\ LSL X12, X10, #3 // i*8 \\ ADD X13, X9, X12 // base address (a) + i*8 \\ LDUR X14, [X13, #0] // X14 = a[i] \\ ADD X11, X11, X14 // result = result + a[i]
```

```
result = a[i] \\ i = i + 1 \\ result = result + a[i] \\ ADDIX13, X9, X12 // base address (a) + i*8 \\ LDUR X11, [X13, #0] // result = a[i] \\ ADDIX10, X10, #1 // i = i+1 \\ LSL X12, X10, #3 // i*8 \\ ADD X13, X9, X12 // base address (a) + i*8 \\ LDUR X14, [X13, #0] // X14 = a[i] \\ ADD X11, X11, X14 // result = result + a[i]
```

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*

```
B L1

ADD X10, X11, X12 //Skipped

L1: SUB X10, X11, X12
```

B Exit

ADD X10, X11, X12 //Skipped

Exit: SUB X10, X11, X12

Think of it as a name for an instruction
Branch name can be anything,
representation is followed by a colon.
EXIT: (is just another label, not a command)

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;
```

CBZ X9, L2

L1: *ADD X10, X11, X12* Skipped

L2: *SUB X10, X11, X12*

Register X9 **0**

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

Register X9 10

L1: *ADD X10, X11, X12* Skipped

L2: *SUB X10, X11, X12*

CBNZ X9, L2

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

If
$$(x == 0)$$

 $f = f + 1$

else

$$f = f - 1$$

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

If
$$(x == 0)$$

$$f = f + 1$$

else

$$f = f - 1$$

CBNZ X9, Else

ADDI X10, X10, #1

B Exit

Else: SUBI X10, X10, #1

Exit:

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

Else: SUBI X10, X10, #1

Exit:

CBZ X9, If

SUBI X10, X10, #1

B Exit

If: ADDI X10, X10, #1

Exit:

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

If
$$(x == 1)$$

 $f = f + 1$

else

$$f = f - 1$$

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

If
$$(x == 1)$$

$$f = f + 1$$

else

$$f = f - 1$$

SUBI X11, X9, #1

CBNZ X11, Else

ADDI X10, X10, #1

B Exit

Else: SUBI X10, X10, #1

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
```

• C code:

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

k in x24

• C code:

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

k in x24

Loop:

B Loop

• C code:

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

k in x24

Loop: ADDI X24, X24, #1

B Loop

• 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:

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

Condition codes/flags

To do the comparison

If
$$(i < j)$$

. . .

) (° ').

LEGv8 code:

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

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
                   // conditional branch
     B.LE Exit
     ADDI X22,X22,#1
Exit:
```

• C code:

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

k in x24

• 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:

• C code:

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

k in x24, and is a signed number

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

• 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 XZR, X24, #10

B.GT Exit

B Loop

Exit:

Final Exam Review

- Chapter 1:
 - Performance
 - CPU Execution time
 - CPI
 - Amdahl's Law
- Chapter 2:
 - Number System
 - Load/Store data from/in memory
 - Assembly language
- Chapter 3:
 - Overflow
 - IEEE 754 representation

Adding 64-bit numbers

Subtraction

• Subtracting 6_{ten} from 7_{ten} directly

- - Subtracting 6_{ten} from 7_{ten} using two's complement. 7 + (-6)