Write a LEGv8 assembly program that implements the following (assume i is in register X9 and the base address of A is in X10 and that the elements of A are long integers [64 bits]):

i = A[10];

A[10] = A[11];

A[11] = i;

LDUR X9, [X10, #80]

LDUR X11, [X10, #88]

STUR X11, [X10, #80]

STUR X9, [X10, #88]

Write the LEGv8 assembly for the following code snippet (assuming the base address of A is in X9, the base address of B is in X10, i is in X11 and j is in X12 and that A and B store 64-bit integer values):

A[i] = B[j];

LSL X13, X12, #3

Compute offset of element j putting the result in register X13

ADD X13, X13, X10

Compute absolute address of B, element j putting the result in register X13

LDUR X13, [X13, #0]

Move B, element j from memory into register X13

LSL X14, X11, #3

Compute offset of element i putting the result in register X14

ADD X14, X14, X9

Compute absolute address of A, element i putting the result in register X14

STUR X13, [X14, #0]

Move the value from register X13 into A, element i in memory

f = g – A[B[4]];

What is the corresponding LEGv8 assembly code? Assume that the variables f, g, and h are assigned to registers X9, X10, and X11 respectively. Assume that the base address of the arrays A and B are in registers X12 and X13, respectively.

LDUR X13, [X13, #32]

LSL X13, X13, #3

ADD X13, X13, X12

LDUR X13, [X13, #0]

SUB X9, X10, X13

1010 1101 0001 0000 0000 0000 0000 0010two

For the 32-bit pattern above, what decimal number does it represent, assuming that it is a two’s complement (signed) integer?

-1391460350.0000

1010 1101 0001 0000 0000 0000 0000 0010two

For the 32-bit pattern above, what decimal number does it represent, assuming that it is an unsigned integer?

2903506946.0000

1010 1101 0001 0000 0000 0000 0000 0010two

For the 32-bit pattern above, what is the hexadecimal representation?

0XAD100002

In the following problems, use the values listed below for registers $t0 and $t1.

X9 = 0x55555555

X10 = 0x12345678

Part 1

LSL X11, X9, #4

EOR X11, X11, X10

What is the value of X11 after the previous sequence of instructions (in hexadecimal)?

0X547610328

Part 2

LSL X11, X9, #4

ANDI X11, X11, #-1

What is the value of X11 after the previous sequence of instructions (in hexadecimal)?

0X555555550

Part 3

LSR X11, X9, #3

ANDI X11, X11, #0xFFEF

What is the value of X11 after the previous sequence of instructions (in hexadecimal)?

0XAAAA

How many byes are needed to store a value of type uint32\_t?

4.0000

How many bits are needed to store a value of type double?

64.0000

1 second = 1,000 milliseconds = 1,000,000,000

A computer supports instructions in the following categories: Memory (2 cycles), Integer Arithmetic (3 cycles), Branching (5 cycles), Floating Point Arithmetic (6 cycles).

Program A has the following instruction counts:

Memory: 100 instructions

Integer Arithmetic: 500 instructions

Branching: 50 instructions

Floating Point Arithmetic: 200 instructions

What is the average CPI for Program A (2 decimal places)?

3.7100

What is the length of a clock cycle in a 2 GHz machine, in picoseconds?

500ps

Amdahl’s Law: The opportunity for improvement is affected by how much time an event consumes. It’s a quantitative law of diminishing returns.

Relative Performance: Performance[x]/Performance[y] = Execution time[y]/Execution[x]

“Static” refers to the code in memory or on the filesystem

“Dynamic” refers to what happens at runtime

Response time: How long it takes to do a task.

Throughput: Total work done per unit time.

1 kilobyte (KB) → 1024 bytes (210 bytes)

1 megabyte (MB) → 1024 KB (220 bytes)

1 gigabyte (GB) → 1024 MB (230 bytes)

X0 – X7: procedure arguments/results

X8: indirect result location register

X9 – X15: temporaries

X16 – X17 (IP0 – IP1): may be used by linker as a scratch register, other times as temporary register

X18: platform register for platform independent code; otherwise a temporary register

X19 – X27: saved

X28 (SP): stack pointer

X29 (FP): frame pointer

X30 (LR): link register (return address)

XZR (register 31): the constant value 0

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0000 | 4 | 0100 | 8 | 1000 | c | 1100 |
| 1 | 0001 | 5 | 0101 | 9 | 1001 | d | 1101 |
| 2 | 0010 | 6 | 0110 | a | 1010 | e | 1110 |
| 3 | 0011 | 7 | 0111 | b | 1011 | f | 1111 |

ASCII: 128 characters

Conditional Branches:

B.EQ (equal)

B.NE (not equal)

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)

Clock Cycles = Instruction Count \* Cycles per Instruction

CPU Time = Instruction Count \* CPI \* Clock Cycle Time = (Instruction Count \* CPI)/Clock Rate

What is the decimal value of the following signed 8-bit binary: (~0x51 + 1)?

-81

What is the decimal value of (75 >> 2) ?

18

Write the LEGv8 assembly for the following code snippet. One trick is to figure out how to get the large constant in a register when the ADDI instruction only allows 12-bit constants. Assume a is stored in register X9.

if(a & 0xA0000 != 0){

a = ~a + 1;}

ADDI X10, XZR, #0XA

Set the value of X10 to the constant 0xA

LSL X10, X10, #16

Shift the value in X10 appropriately to get the constant from the code snippet, saving back in X10

SUB X10, X9, X10

Compute the difference between 'a' and the constant in X10, storing the result in X10

CBZ X10, EXIT

If the difference computed by the previous instruction is zero, branch to the label EXIT

SUBI X10, XZR, #1

Get the constant -1 into the X10 register using SUBI

EOR X9, X9, X10

Flip all the bits of 'a' putting the result into X9

ADDI X9, X9, #1

Increment 'a' by 1

EXIT

What is the LEVv8 assembly for the following code snippet: a = 0;

for(i=0;i<5;i++){a=a+B[i];}

Assume 'a' is in X9 and the base address of B is in X10. 'i' is in X11 and B stores 64-bit integers.

ADDI X9, XZR, #0

Initialize 'a' to zero.

ADDI X11, XZR, #0

Initialize 'i' to zero.

LOOP:

SUBIS X12, X11, #5

Subtract 5 from 'i' storing the result in X12 and setting the condition code.

B.GE EXIT

Conditional branch to EXIT if the previous result was greater than or equal to zero (signed).

LSL X12, X11, #3

Multiply 'i' appropriately to compute the offset putting the result in X12

ADD X12, X12, X10

Add the offset of 'i' to the base address of B putting the result in X12

LDUR X12, [X12, #0]

Load the value at B index i into X12

ADD X9, X9, X12

Increment 'a' by the value of B index i.

ADDI X11, X11, #1

Increment 'i' by 1.

B LOOP

Unconditionally branch to the label LOOP

EXIT:

Write the LEGv8 assembly code to call a procedure with the following prototype:

long long int my\_procedure(long long int a, long long int b, long long int c);

Before the call, the values to be passed in are in X19, X20 and X21 and after the call, the return value should be stored in X22.

Just do the basic call, don't worry about storing values of registers X19-X22 on the stack.

ADD X0, X19, XZR

Copy the value of 'a' into the appropriate procedure parameter register.

ADD X1, X20, XZR

Copy the value of 'b' into the appropriate procedure parameter register.

ADD X2, X21, XZR

Copy the value of 'c' into the appropriate procedure parameter register.

BL my\_procedure

Make the call to 'my\_procedure'.

ADD X22, XZR, X0

Copy the returned value into the appropriate register.

Write the LEGv8 assembly code for the following loop. This code compares two character arrays and sets the variable different to 1 if they are not the same. A and B are arrays of characters (unsigned, 8 bits per element)

int different = 0; int index = 0;

while(1= =1){if(A[index]!= B[index])

{different = 1; break;}

if(A[index]= = ‘\0’) {break;}

index++;}

Assume different is in X9, the base address of A is in X10 and the base address of B is in X11; index should go in X12.

ADDI X9, XZR, #0

Initialize the value of 'difference'.

ADDI X12, XZR, #0

Initialize the value of 'index'.

LOOP:

ADD X13, X10, X12

Computer absolute address of A\_i storing the result in X13.

ADD X14, X11, X12

Compute absolute address of B\_i storing the result in X14

LDURB X13, [X13, #0]

Load A\_i into X13.

LDURB X14, [X14, #0]

Load B\_i into X14.

SUB X15, X13, X14

Compute the difference of A\_i and B\_i storing in X15

CBZ X15, SECONDIF

If the two values are the same, branch to label SECONDIF.

ADDI X9, XZR, #1

Set 'different' to 1.

B EXIT

Branch to the label EXIT

SECONDIF:

CBZ X13, EXIT

Branch to EXIT if A\_i is equal to zero.

ADDI X12, X12, #1

Increment 'index' by 1.

B LOOP

Go back to the LOOP label.

EXIT:

Write a leaf procedure with the following prototype:

long long int strlen(char buffer[]);

It should return the number of characters in the character array buffer (by starting at index zero and looking at the characters in the array. The "sentinel" character at the end is '\0' which is also the number zero and this should not be counted).

Compute the absolute address of buffer\_i storing in X10​

STRLEN:

ADDI X9, XZR, #0

Initialize counter register X9 to zero

STRLEN\_LOOP:

ADD X10, X0, X9

Compute the absolute address of buffer\_i storing in X10.

LDURB X10, [X10, #0]

Load buffer\_i into X10.

CBZ X10, STRLEN\_RETURN

Branch to STRLEN\_RETURN label if buffer\_i is zero.

ADDI X9, X9, #1

Increment counter register by 1.

B STRLEN\_LOOP

Branch to the STRLEN\_LOOP label.

STRLEN\_RETURN

ADD X0, X9, XZR

Copy value of the counter to X0.

BR LR

Return to the caller.

Write the stack manipulation instructions for a leaf procedure that must store X0 and X19 onto the stack.

SUBI SP, SP, #16

Allocate storage on the stack for two registers.

STUR X0, [SP, #8]

Store X0 in the second element relative to the stack pointer.

STUR X19, [SP, #0]

Store X19 in the first element relative to the stack pointer.

Write the ending stack manipulation instructions for a non-leaf procedure that saved X0 and X19 and X20 (in that order of offset), and also LR.

LDUR X0, [SP, #0]

Load X0 from the stack.

LDUR X19, [SP, #8]

Load X19 from the stack.

LDUR X20, [SP, #16]

Load x20 from the stack.

LDUR LR, [SP, #24]

Load LR from the stack.

ADDI SP, SP, #32

Restore the stack pointer.

Implement the following in LEGv8 assembly. Assume that "my\_global" is stored at offset 8 (bytes) from the start of global memory. Further assume, the address of the start of global memory is stored in register X29, and is available throughout the code.

uint16\_t my\_global = 100;

uint64\_t non\_leaf()

{uint64\_t x = 10;

return my\_function(x, my\_global);}

uint64\_t my\_function(uint64\_t x, uint64\_t y){return x - y;}

Store the return address onto the stack​

init\_globals: 'init\_globals' is a procedure called at the beginning of the program. Its job is to initialize global variables.

ADDI X9, XZR, #100

Initialize register X9 with the constant 100

STURH X9, [X29, #8]

Store the value of X9 into the memory dedicated to 'my\_global'

BR LR

Return

my\_function:

SUB X0, X0, X1

Subtract 'y' from 'x' and put the result in X0

BR LR

Return

non\_global

SUBI SP, SP, #8

Allocate room on the stack for the return address

STUR LR, [SP, #0]

Store the return address onto the stack

ADDI X0, XZR, #10

Use X0 for 'x' and initialize it to 10

LDURH X1, [X29, #8]

Load the value of 'my\_global' into X1

BL my\_function

Call 'my\_function'

LDUR LR, [SP, #0]

Load the return address from the stack

ADDI SP, SP, #8

De-allocate the stack memory

BR LR

Return