Review Problem 6

- Register X0 has the address of a 3 integer array. Set X15 to 1 if the array is sorted (smallest to largest), 0 otherwise.

```
ADD   X15, X31, X31     //initialize to 0
LDUR  x1, [x0, #0]      //array[0]
LDUR  x2, [x0, #8]      // array[1]
LDUR  x3 ,[x0, #16]     // array[2]
CMP   x1, x2            //
B.GET UNSORTED
CMP   x2, x3
B.GT  UNSORTED
ADDI  X15, x31, #1      // set to 1 - sorted!
UNSORTED:
```

Labels

Labels specify the address of the corresponding instruction

Programmer doesn't have to count line numbers

Insertion of instructions doesn't require changing entire code

```
        // X0 = N, X1 = sum, X2 = I
-6 -3   ADD X1, X31, X31    // sum = 0
-5 -2   ADD X2, X31, X31    // I = 0
        TOP:
-4 -1   CMP X2, X0          // Check I vs N
-3  0   B.GE END +4         // end when !(I<N)
-2 +1   ADD X1, X1, X2      // sum += I
-1 +2   ADDI X2, X2, #1     // I++
 0 +3   B TOP -4            // next iteration
        END:
+1 +4
```

Notes:

Branches are PC-relative

PC = PC + 4*(BranchOffset)

BranchOffset positive -> branch downward. Negative -> branch upward.

Labels Example

Normally: PC = PC + 4

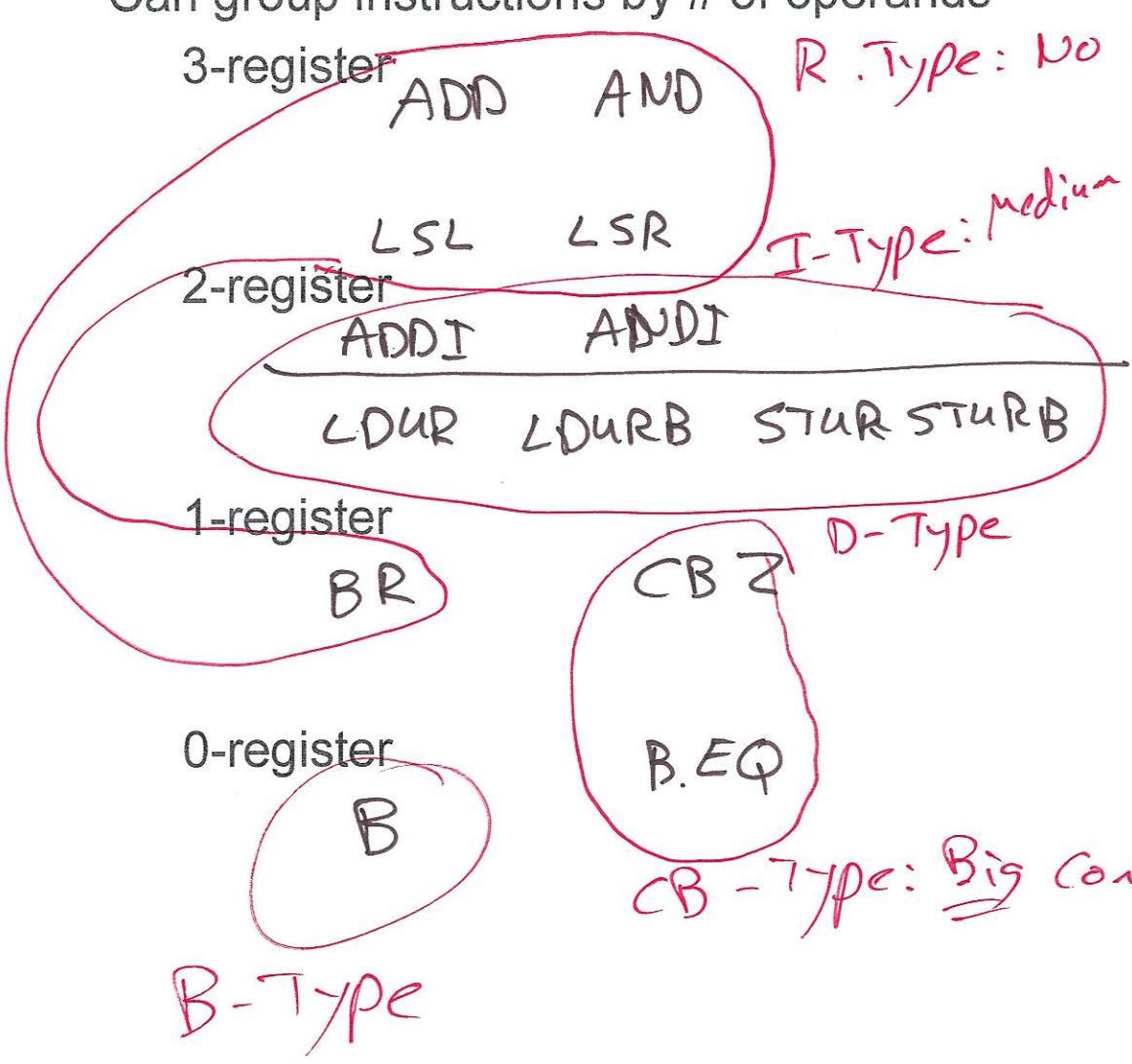
Compute the value of the labels in the code below.

Branches: PC = PC + 4*(BranchOffset)

```
// Program starts at address 100
    LDUR   X0, [X31, #100]
LOOP:
    LDURB  X1, [X0, #0]
    CBZ    X1, END      +9
    CMPI   X1, #97
    B.LT   NEXT         +5
    CMPI   X1, #122
    B.GT   NEXT         +3
    SUBI   X1, X1, #32
    STURB  X1, [X0, #0]
NEXT:
    ADDI   X0, X0, 1
    B      LOOP         -9
END:
```

Instruction Types

Can group instructions by # of operands

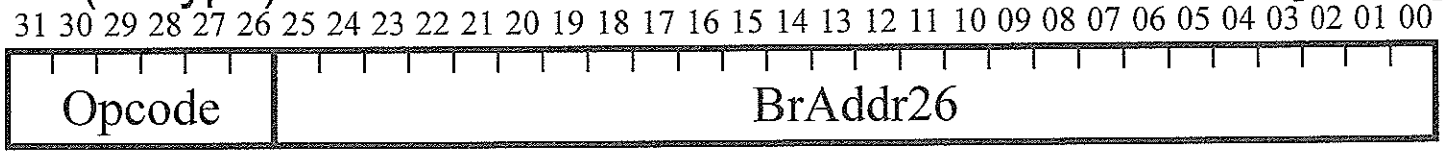



```
ADD X0, X1, X2
ADDI X0, X1, #100
AND X0, X1, X2
ANDI X0, X1, #7
LSL X0, X1, #4
LSR X0, X1, #2
LDUR X0, [X1, #14]
LDURB X0, [X1, #14]
STUR X0, [X1, #14]
STURB X0, [X1, #14]
B START
BR X30
CBZ X0, FOO
B.EQ DEST
```

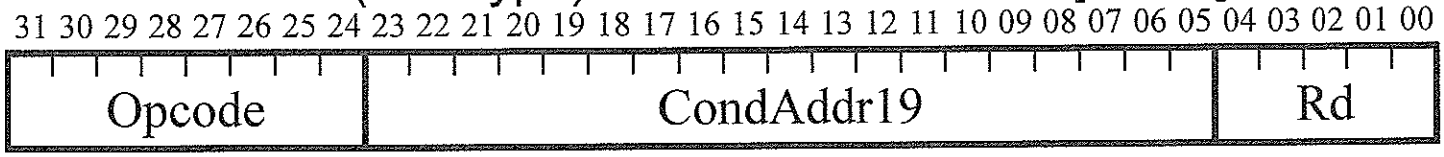
Instruction Formats

All instructions encoded in 32 bits (operation + operands/immediates)

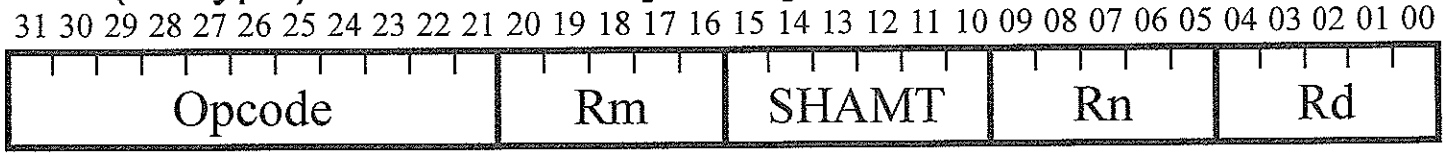
Branch (B-Type) Instr[31:21] = 0A0-0BF



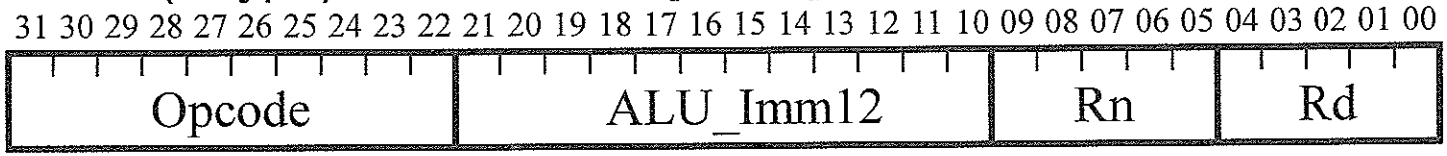
Conditional Branch (CB-Type) Instr[31:21] = 2A0-2A7, 5A0-5AF



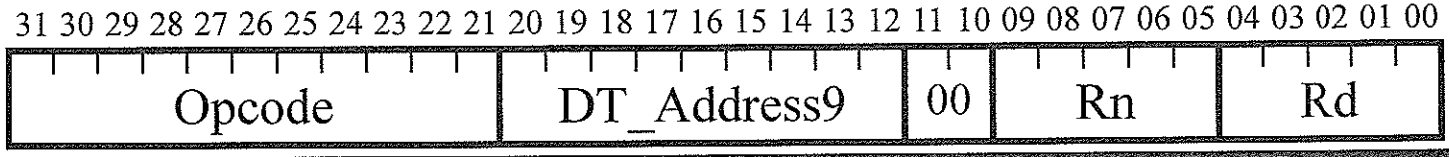
Register (R-Type) Instr[31:21] = 450-458, 4D6-558, 650-658, 69A-758



Immediate (I-Type) Instr[31:21] = 488-491, 588-591, 688-691, 788-791

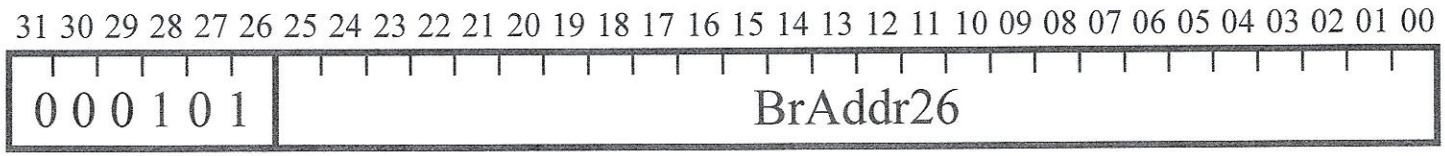


Memory (D-Type) Instr[31:21] = 1C0-1C2, 7C0-7C2



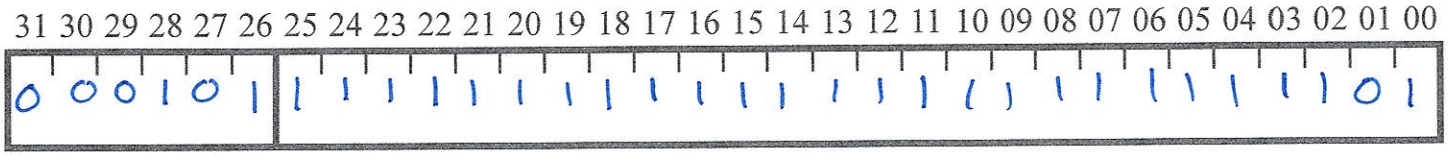
B-Type

Used for unconditional branches



0x05: B

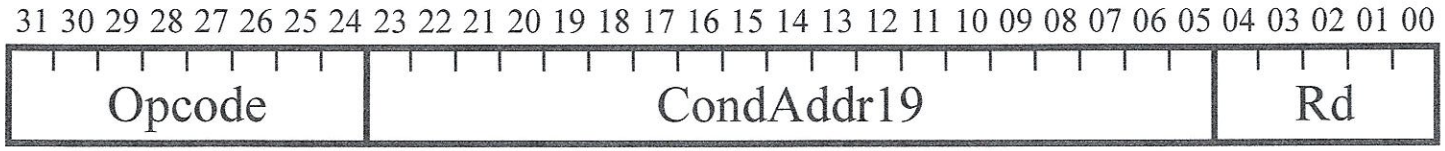
```
B -3    // PC = PC + 4*-3
```



B: 05_{16} -3

CB-Type

Used for conditional branches

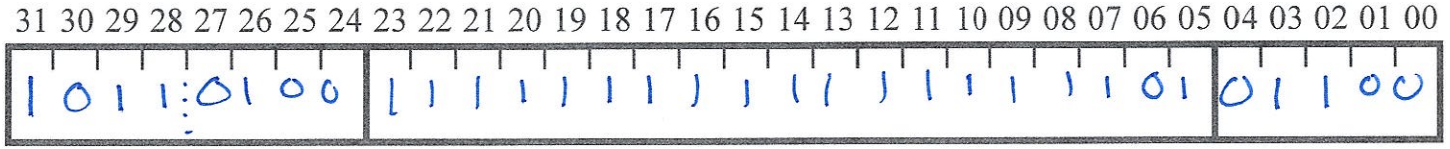


Reg or Cond. Code

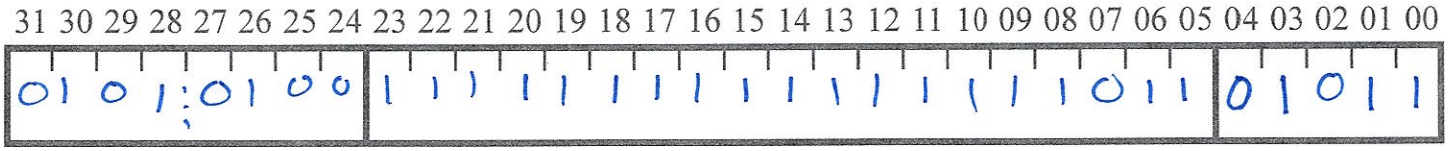
0x54: B.cond
0xB4: CBZ
0xB5: CBNZ

Condition Codes
0x00: EQ (==)
0x01: NE (!=)
0x0A: GE (>=)
0x0B: LT (<)
0x0C: GT (>)
0x0D: LE (<=)

```
CBZ X12, -3        // if(X12==0) PC = PC + 4*-3
```

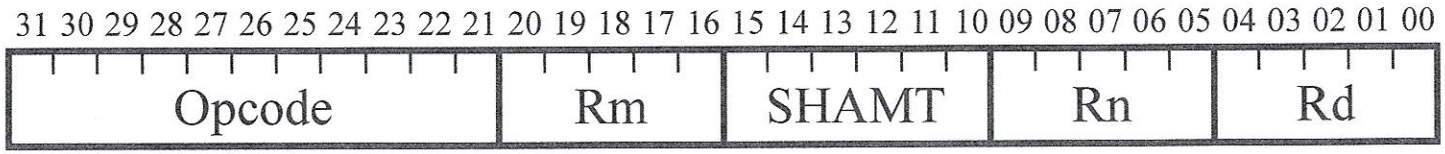



```
B.LT -5            // if (lessThan) PC = PC + 4*-5
```



R-Type

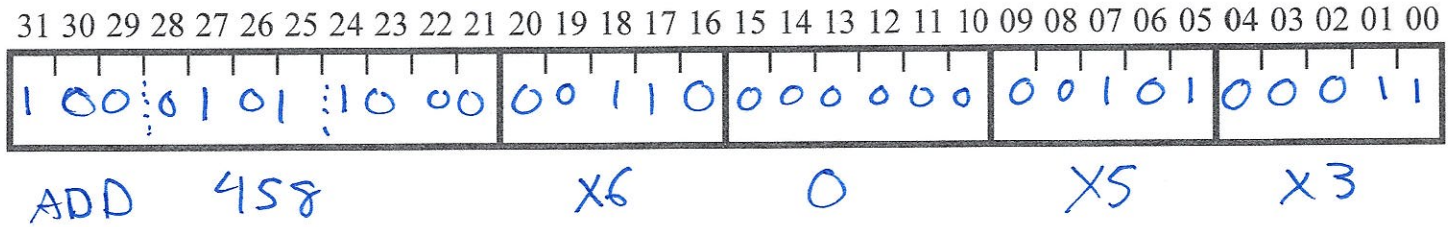
Used for 3 register ALU operations and shift



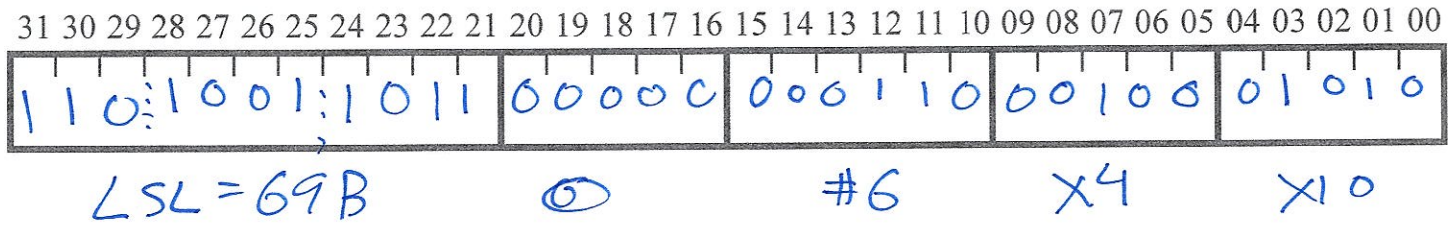
Op2 (0 for shift) | Shift amount (0 for non-shift) | Op1 | Dest

0x450: AND
0x458: ADD
0x4D6: SDIV, shamt=02
0x4D8: MUL, shamt=1F
0x550: ORR
0x558: ADDS
0x650: EOR
0x658: SUB
0x69A: LSR
0x69B: LSL
0x6B0: BR, rest all 0's but Rd
0x750: ANDS
0x758: SUBS

```
ADD     X3, X5, X6      // X3 = X5+X6
```

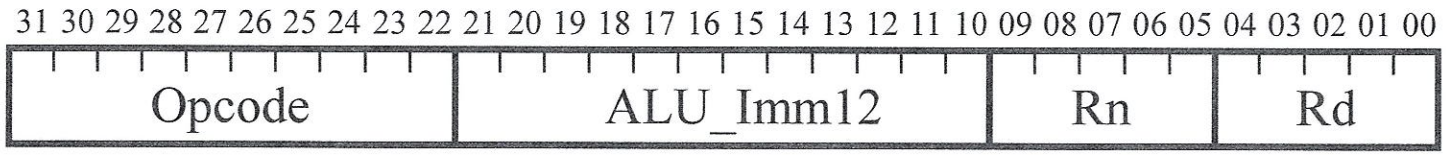



```
LSL     X10, X4, #6     // X10 = X4<<6
```



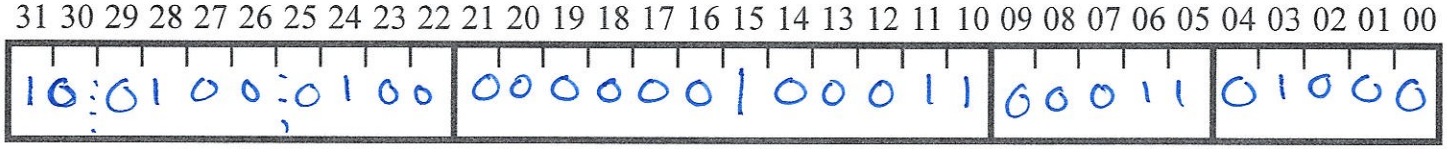
I-Type

Used for 2 register & 1 constant ALU operations



0x244: ADDI
0x248: ANDI
0x164: ADDIS
0x168: ORRI
0x344: SUBI
0x348: EORI
0x2C4: SUBIS
0x2C8: ANDIS

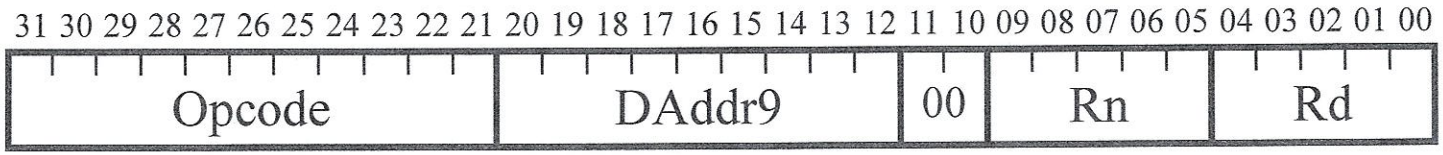
```
ADDI     X8, X3, #35     // X8 = X3+35
```



ADDI 244 #35 X3 X8

D-Type

Used for memory accesses



0x1C0: STURB
0x1C2: LDURB
0x7C0: STUR
0x7C2: LDUR

```
LDUR      X6, [X15, #12]      // X6 = Memory[X15+12]
```