Outline

- Loops

A Loop with a variable array index

```
Loop:   g = g + A[i];
        i = i + j;
        If (i != h) go to Loop
```

Variables g, h, i, and j associated with $s1 → $s4

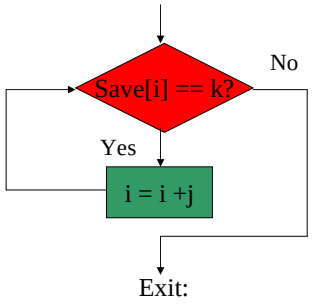
Array base is in $s5. What is the MIPS assembly code?

```
Loop:   add $t1, $s3, $s3       # $t1 ← 2 *i
        add $t1, $t1, $t1       # $t1 ← 4 *i
        add $t1, $t1, $s5       # $t1 ← address of A[i]
        lw $t0, 0($t1)          # $t0 ← A[i]
        add $s1, $s1, $t0       # g ← g + A[i]
        add $s3, $s3, $s4       # i = i+ j
        bne $s3, $s2, Loop      # go to Loop if i != h
```


A While Loop 1/2

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

i, j, and k associated with $s3 → $s5. Base of array save is in $s6. What is MIPS Assembly code?



Exit:

A less than test

if (a < b) go to Less

a, b associated with $s0, and $s1, respectively. What is the MIPS assembly code?

- To test for less than we introduce a new instruction

  slt $t0, $s0, $s1 # slt is set on less than

  Register $t0 is set to 1 if the value in register $s0 is less than the value in register $s1, otherwise, register $t0 is set to 0

- Register $zero always contains zero
- Comparing $t0 to $zero gives us the effect of branching if a less than b
- Combining slt with bne implements branch on less than

```
        slt $t0, $s0, $s1       # $t0 ← 1 if a < b, otherwise $t0 ← 0
        bne $t0, $zero, Less    # go to Less if $t0 != 0
        ……
Less:
```

Case/Switch Statement1/3

•Switch statement allows the programmer to select one of many alternatives depending on a single value

•Can be implemented as a series of if –then-else statements (not efficient)

•Alternatively (or more efficiently)

➢First instruction within each case sequence is assigned a label (the label holds the address of the first instruction in the sequence)

➢These addresses are stored in a jump table in memory

•Use new instruction

jr register (jump register)

Unconditional jump to the address specified in register

•Program loads appropriate entry from jump table into a register, then jump to proper address using a jump register

Case/Switch Statement2/3

```
switch (k) {
        case 0:   f = i + j; break;
        case 1:   f = g + h; break;
        case 2:   f = g - h; break;
        case 3:   f = i – j; break;
}
```

Variables f through k associated with $s0 → $s5. Register $t2 contains 4. Register $t4 contains address of jump table in memory. MIPS Assembly code?

Address of instruction f = i + j → label L0

Address of instruction f = g+ h → label L1

Address of instruction f = g - h → label L2

Address of instruction f = i - j → label L3

$t4: 1024

	Jump table stored in memory
1036	L3
1032	L2
1028	L1
1024	L0

Case/Switch Statement3/3

```
        slt $t3, $s5, $zero      # test if k < 0
        bne $t3, $zero, Exit     # if k < 0 go to Exit
        slt $t3,$s5,$t2          # test if k < 4
        beq $t3, $zero, Exit     # if k >= 4 go to Exit
        add $t1,$s5,$s5          # $t1 ← 2 *k (Why?)
        add $t1,$t1,$t1          # $t1 ← 4 *k
        add $t1,$t1,$t4          # $t1 ← address of JumpTable[k]
        lw $t0,0($t1)            # $t0 ← JumpTable[k]
        jr $t0                   # jump to address found in register $t0
L0:     add $s0, $s3, $s4        # (k = 0) then f = i +j
        j Exit                   # break (go to Exit)
L1:     add $s0,$s1,$s2          #(k = 1) then f = g+h
        j Exit                   # break (go to Exit)
L2:     sub $s0,$s1,$s2          #(k=2) then f = g-h
        j Exit
L3:     sub $s0,$s3,$s4          #(k=3) then f = i- j
Exit:                            # End of switch statement
```


What we know so far

•Fig. 3.9 page 131

•instruction format for unconditional jump j (J-format)

<opcode (6 bits), target address (26 bits)> (more on target address in section 3.8)

•The jr instruction → (R-format)

opcode = 0, funct = 8, rs is jump register

•The slt instruction → (R-format)

Opcode = 0, funct = 42, rs, rd, rt

•The beq, bne instructions → (I-format)

•$zero always contains the value 0

•The instruction beq $s1, $s2, 100 is an I-format instruction. The resulting machine code

•Op| rs | rt | address

4 | 17 | 18 | 25

•Why the value in address field is 25 and not 100? Will discuss this when we cover section 3.8