Decision Making and arrays
if and loops

- Need to do one of two things
  - if : do next instruction or skip to another (else)
  - loop: continue in loop or exit loop
Decision instructions in ARMv8

- **Simple instructions:**
  - `cbz x9, label`
    » Conditional branch on zero
    » If value in X9 equals zero, go to instruction at label
    » otherwise, execute next instruction in order
  - `cbnz x9, label`
    » Conditional branch if not zero
    » If value in X9 does not equal zero, go to instruction at label
    » otherwise, execute next instruction in order
if statement in assembly

if(a == b)  // a in X19, b in X20
  {c = 1}   // c in X21
// next instruction

sub X9, X19, X20
cbnz X9, endif
mov X21, #1
endif:  // next instruction
if (a == b)  // a in X19, b in X20
    {c = 1}  // c in X21
else
    {c = 0}
// next instruction
if – else in assembly

sub X9, X19, X20
cbnz X9, else
mov X21, 1
b endif

else:
mov X21, 0
endif: # next instruction
What about other comparisons?

- $a < b$
- $a \leq b$
- etc…
Conditional branches based on flags

- Flags set by instructions
  - set by arithmetic/logical instructions
  - indicate facts about previous result
  - four bits
    - negative (N) result was negative
    - zero (Z) result was zero
    - overflow (V) result had overflow
    - result (C) result had a carry out of most significant bit, or borrow into most significant bit
Conditional branches based on flags

- **Most commonly used with subs**
  - subtract and set flags
  - subs XZR, X9, X10
    - X9 – X10, set flags, can’t write to XZR
  - Condition branches on relation between X9, X10
    - b.lt branch if less than; b.le branch if less than or equal
    - b.gt, (b.ge) branch if greater than (or equal)
  - Unsigned conditions
    - b.lo, (b.ls) branch if lower (or same)
    - b.hi, (b.hs) branch if higher (or same)
Conditional branches based on flags

- Can test flags directly, when they are set by one of a limited set of operations, (append s to instruction to indicate that flags will be set, e.g. adds)
  - b.m branch on minus \(N = 1\)
  - b.pl branch on plus \(N = 0\)
  - b.vs branch if overflow \(V = 1\)
  - b.vc branch if overflow clear \(V = 0\)
if (a < b)    // a in X19, b in X20
    {c = 1}    // c in X21

// next instruction
subs XZR, X19, X20
b.ge endif
mov X21, 1
endif:  # next instruction
while loop

while (condition)
{
  loop body
}

Each time at the top of the loop, check the condition. If true, continue the loop. At the end of the loop, go back to check the condition again.
while loop example - c

while(count < 10)
{
    // do something
    count ++;
}

// next instruction
while loop example - assembly

// Assume count is in register X10, 
// and register X19 contains 10

loop:      subs XZR, X10, X19
        b.?? loopend
        # do something
        add X10, X10, 1
        b loop

loopend:   # next instruction
while loop example - assembly

// Assume count is in register X10, // and register X19 contains 10

loop:
    subs XZR, X10, X19
    b.ge loopenend
    # do something
    add X10, X10, 1
    b loop

loopenend:
    # next instruction
for loop example - c

for (cnt = 5, cnt >= 0, cnt--)
{
    // do something
}

// next instruction
for loop example - assembly

mov   X10, #5

loop:
subs  XZR, X10, #0
b.lt  loopend
# do something
sub   X10, X10, #1
b    loop

loopend:
# next instruction
Arrays

- Base address of array A is address of A[0]
  - Usually stored in a register, like X20
  - Length also usually stored in a register, X19
- Typical loop uses loop counter, perhaps k, to access array elements, A[k]
- Arrays of integers, each element is 64 bits or 8 bytes
  - So array addresses increment by 8
Array in memory

10000  Holds A[0]  base address
10008  Holds A[1]
10016  Holds A[2]
...
??     Holds A[k]
Accessing arrays

Assume base address (A[0]) is in X20
Assume index, k, is in X9
Then address of A[k] is 8*X9 + X20

```
lsl   X11, X9, 3       // left shift by 3 = *8
add   X11, X11, X20    // X11 is addr of A[k]
ldur  X12, [X11, 0]    // X12 holds value of A[k]
```
For loop example

```c
sum = 0;
for (k = 0; k < length; k++)
    sum = sum + A[k];
```

Assume sum is in X10, k in X9
base address for A is already in X20,
length is already in X19
Assembly code

```assembly
classess X10, XZ, R XZR // sum = 0
add XR9, XZ,R, XZ R // k = 0
add X9, XZ R, XZ R

for:      subs XZ R, X9, X19 // is k < length
b.ge endfor // if not end loop
lsl X11, X9, 3 // X 11= 8*k
add X11, X11, X20 // X11 is addr A[k]
ldur X12, [X11, 0] // X12 = A[k]
add X10, X10, X12 // sum = sum + A[k]
add X9, X9, 1 // k++
b for
endfor: <next instruction>
```