10	20	30	4
5:54	6:08		

Flow control constructs

- IF

```
if(a>b)    →    CMP  a,b        →    CMP  a,b
   S1           BGT  Dothis          BLE  Skip
                B    Skip            {S1}
        Dothis       {S1}
        Skip          .           Skip
```

reversion : switch a>b to a<b to skip

- ## Conditional Execution

eg. MOVEQ ≡ mov if equal

- IF-ELSE

```
if(a==3)            CMP  a,#3          CMP  a,#3
  {S1}      →       BEQ  DoIf          BNE  DoElse
else                  {S2}        →      {S1}
  {S2}              B  Skip              B  Skip
               Doif    {S1}        DoElse  {S2}
               Skip     :          Skip     :

                         CMP  a,#0
                         MOVEQ  r1,#3
                         MOVNE  r1,#5
```

COMPOUND CONDITIONS

→ if ((a == b) && (b>0)) {S1}   AND : L→R

↓

if (a != b) then skip   least-likely
if (b<=0) then Skip
{S1}
Skip

↓

```
         CMP    R1, R2    (R1←a, R2←b)   } a>b and b>0
T= a>b   not ex → CMPEQ  R2, #0   -       }   gives
↳ exc. → XXXGT  XXXX ; S1.                }
```

↓

```
CMP    R1, R0
BNE    Skip          ☆☆
CMP    R2, #0          ✓
XXXGT  XXXX .
```

put most likey here

→ if ((a==1) || (a==2)}

a==1 then DoIf → if half, Halway to truth
a!=2 then skip

```
   CMP     R1, #1
   BEQ     doIF
   CMPNE       R1, #2
doif xxx EQ    XXXX,
```

BRANCHLESS LOGIC

```
if ((X&2) == 2)          AND  R1, R1, #0x02
   x = true              ROR  R1, R1, #2
 else
   x = false
```

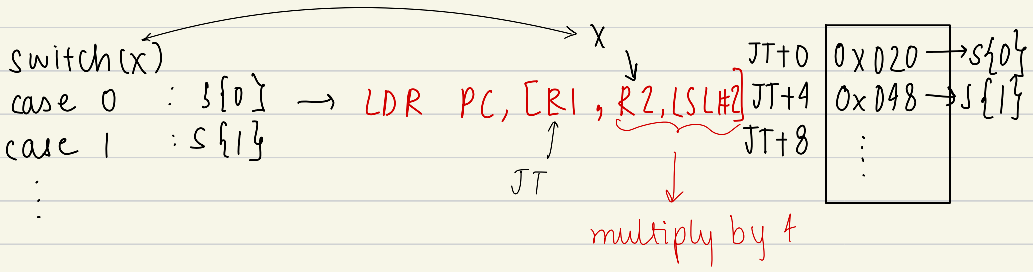
	10	20	30
6:43	6:57	7:07	

· SWITCH

continuous values

Jumptable : has $\overset{\text{starting}}{\uparrow}$ addresses of each executable switch instruction set

var x on which switch is decided = offset.

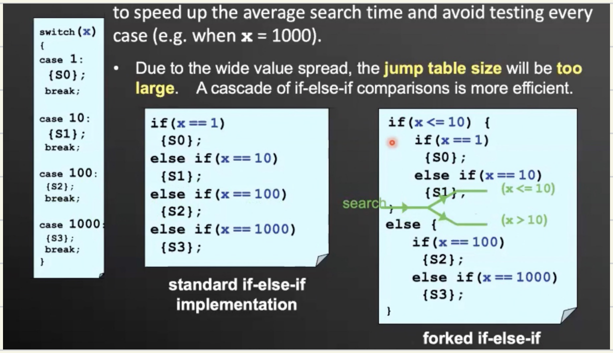


switch(x) → x
case 0 : S{0} → LDR PC, [R1, R2, LSL#2]
case 1 : S{1}
⋮

JT
multiply by 4

JT+0	0x020 → S{0}
JT+4	0x048 → S{1}
JT+8	⋮

· random wide values ⟹ fork algorithm



to speed up the average search time and avoid testing every case (e.g. when **x** = 1000).

- Due to the wide value spread, the **jump table size** will be **too large**. A cascade of if-else-if comparisons is more efficient.

```
switch(x)
{
case 1:
  {S0};
  break;

case 10:
  {S1};
  break;

case 100:
  {S2};
  break;

case 1000:
  {S3};
  break;
}
```

```
if(x==1)
  {S0};
else if(x==10)
  {S1};
else if(x==100)
  {S2};
else if(x==1000)
  {S3};
```

standard if-else-if implementation

```
if(x<=10) {
   if(x==1)
    {S0};
   else if(x==10)
    {S1};
}
else {
   if(x==100)
    {S2};
   else if(x==1000)
    {S3};
}
```

forked if-else-if

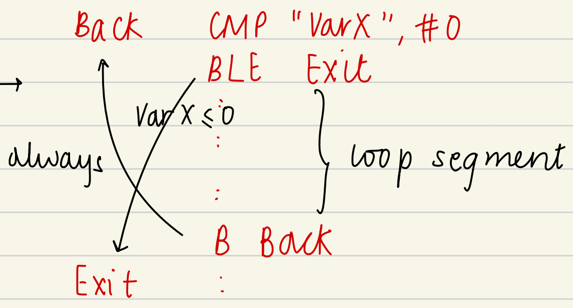
LOOPS

- pretest vs posttest
  - pretest ↓ may never execute loop code ↑ while, for
  - posttest ↳ will execute once ← do while

- WHILE

```
while ( VarX > 0 )
{
      loop seg
}
```

→

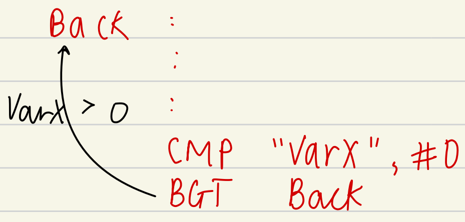


```
Back    CMP  "VarX", #0
        BLE  Exit
         :
         :          } loop segment
        B    Back
Exit     :
```

- DO WHILE

```
DO{
      loop seg
  } while (VarX > 0)
```

→



```
Back    :
        :
        :
        CMP  "VarX", #0
        BGT  Back
```

FOR (pretest)

```
FOR(N=0; N≤5; N++)
{
}
```

Pre implementation

```
        MOV  R0, #0
→ Back  CMP  R0, #5
        BGE  Exit
        :
                      always    N≥5:
        ADD  R0, R0, #1
        B    Back
  Exit  :
```

if count N is not used
⇒ terminate if ≥N
then we can decrement
using post-test and test
for zero.

↓

post implementation

```
      MOV   R0, #5
Back  :
      :
      :
      SUBS  R0, R0, #1
      BNE   Back
```