Branches

R. Ferrero

Politecnico di Torino

Dipartimento di Automatica e Informatica (DAUIN)

Torino - Italy

Unconditional branch

- There are four instructions for unconditional branch:
  - branch `B <label>`
  - branch indirect `BX <Rn>`
  - branch and link `BL <label>`
  - branch indirect with link `BLX <Rn>`
- `BL` and `BLX` save the return address (i.e., the address of the next instruction) in `LR` (`r14`) and they are used to call subroutines.

Infinite loops

- A stand-alone program, without operating system, can not continue beyond the end, otherwise the behavior is unpredictable.
- An infinite loop is added as last instruction:

```
stop B stop
```

or

```
      LDR r1, =stop
stop BX r1
```

LDR Vs ADR with BX

- Instructions are 16 or 32 bits long, so their address is always halfword aligned.
- `BX` requires that the last bit of the register is 1, otherwise a usage fault exception is raised.
- `BX` jumps to the address created by changing the last bit of the register to 0.
- `LDR` sets the last bit to 1 if the label is in a code area, to 0 if the label is in a data area.
- `ADR` and `ADRL` do not change the last bit.

LDR Vs ADR: example

1.

0x000000CC	`LDR r1, =stop`
0x000000CE	`stop BX r1`

Ok: `r1` = 0x000000CF

2.

0x000000CC	`ADRL r1, stop`
0x000000D4	`stop BX r1`

No: `r1` = 0x000000D4

3.

0x000000CC	`ADRL r1, stop`
0x000000D4	`ORR r1, r1, #1`
0x000000D8	`stop BX r1`

Ok: `r1` = 0x000000D9

Branch range

- In the `B` instruction, the opcode is 8 bit and the immediate value is 24 bit.
- Since addresses are halfword-aligned, the immediate value specifies bit 24-1 of the relative address.
- The 25^{th} is for the sign; so the relative address can be $\pm 2^{24}$ byte = $\pm$16 MB.
- `BX` can jump to any 32-bit value = 4 GB.

MOV for unconditional branch

- `B` and `BX` change the value of `PC`.
- Similarly, a jump can be implemented by changing the value of `PC` with `MOV` and `LDR`:
  - ```
    LDR <Rd>, =<label>
    MOV PC, <Rd>
    ```
  - ```
    LDR PC, =<label>
    ```
- `MOV` and `LDR` force the last bit of `PC` to 0.
- `MOV` instead of `BX` is discouraged: the assembler generates a warning.

Conditional branch: B?? and BX??

??	Flags	Meaning
EQ	Z = 1	equal
CS HS	C = 1	unsigned ≥
MI	N = 1	negative
VS	V = 1	overflow
HI	C =1 & Z = 0	unsigned >
GE	N ≥ V	signed ≥
GT	Z =0 or N = V	signed >

??	Flags	Meaning
NE	Z = 0	not equal
CC LO	C = 0	unsigned <
PL	N = 0	positive or 0
VC	V = 0	no overflow
LS	C =0 & Z = 1	unsigned ≤
LT	N ≠ V	signed <
LE	Z =1 or N ≠ V	signed ≤

Example: do you pass the exam?

```
; r0 contains the score of the exam
        CMP r0, #18
        BEQ refuse
        BLO reject
        BHI accept
        ...
refuse  ...     ; study more
reject  ...     ; study much more
accept  ...     ; go on holiday
```

Compare and branch

- Compare and branch if Zero:

  `CBZ <Rn>, <label>`

  jumps to `label` if `Rn` = 0
- Compare and branch if Nonzero:

  `CBNZ <Rn>, <label>`

  jumps to `label` if `Rn` ≠ 0
- `Rn` must be among `r0`-`r7`.
- Only forward branch is possible (4-130 byte).

CBZ-CBNZ Vs conditional branch

- These instructions are almost equivalent:

```
CBZ r0, myLabel
```

```
CMP r0, #0
BEQ myLabel
```

```
CBNZ r0, myLabel
```

```
CMP r0, #0
BNE myLabel
```

- Differences:
  - `CMP` sets the flags, while `CBZ` and `CBNZ` do not.
  - `CBZ` and `CBNZ` jump only forward, range is shorter
  - `CBZ` and `CBNZ` cannot be used within an `IT` block.

While loop

The pseudocode of the while loop is

```
while (r0 != N) {
    ...      //do something
}
```

While loop: implementation

1.
```
        B test
loop    ...         ; do something
test    CMP r0, #N
        BNE loop
```
2.
```
test    CMP r0, #N
        BE exit
        ...         ; do something
        B test
exit
```

While loop with CBZ

If `N` = 0, an alternative implementation using `CBZ` is:

```
loop    CBZ r0, exit
        ...       ; do something
        B loop
exit    ...
```

For loop

The pseudocode of the for loop is

```
for (i = 0; i < N; i++) {
    ...      //do something
}
```

For loop: naive implementation

```
        MOV r0, #0
loop    CMP r0, #N
        BHS exit
        ...         ; do something
        ADD r0, r0, #1
        B loop
exit
```

For loop: optimization

```
        MOV r0, N
loop    ...       ; do something
        SUBS r0, r0, #1
        BNE loop
exit
```

- `CBNZ r0, loop` can not be used instead of `BNE loop` because the branch is backward.

Do-While loop

The pseudocode

```
do {
    ...         //do something
} while (r0 != N);
```

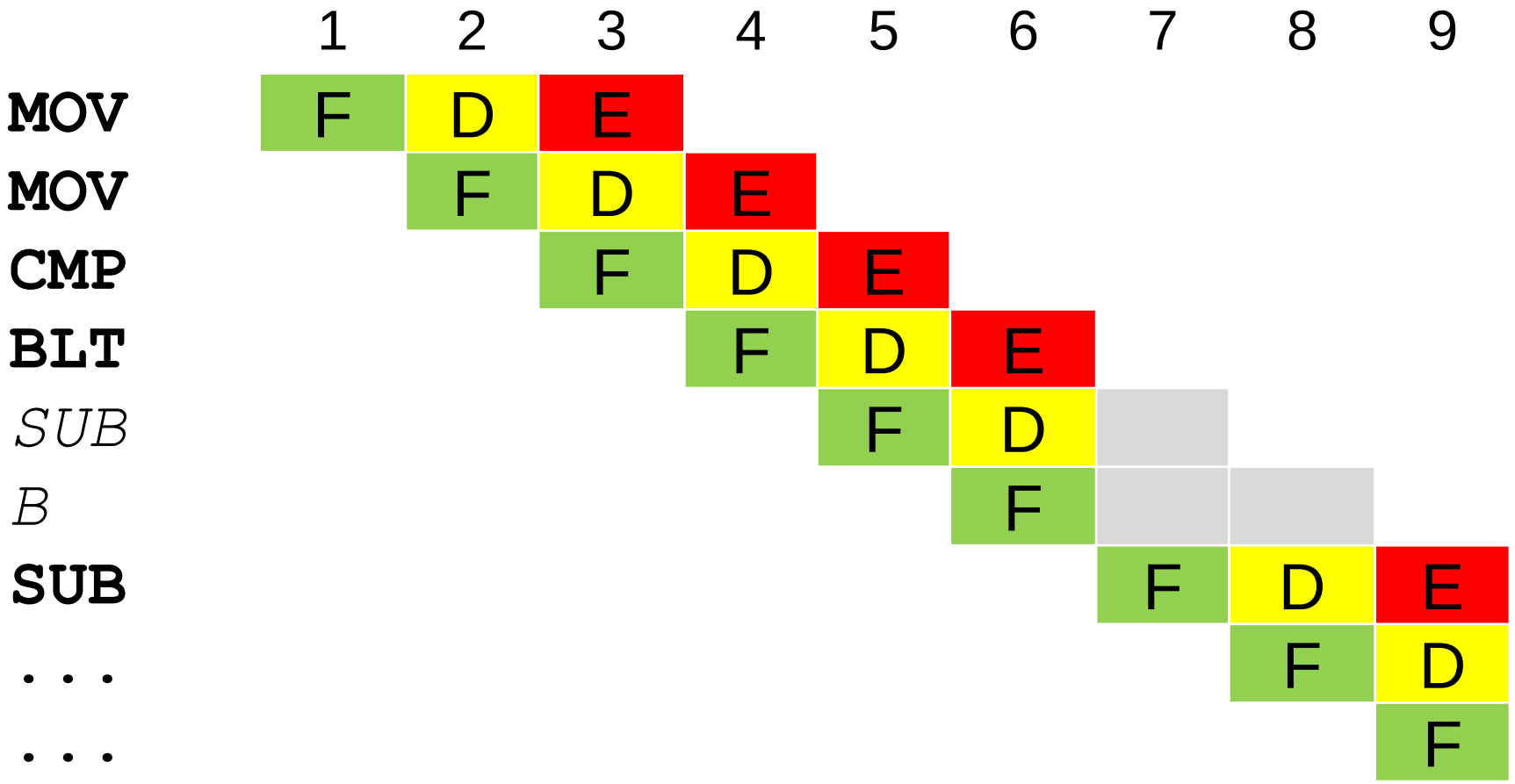
can be implemented as:

```
loop    ...         ; do something
test    CMP r0, #N
        BNE loop
```

Example: absolute value of N - M

```
        MOV r0, #N
        MOV r1, #M
        CMP r0, r1
        BLT neg
        SUB r0, r0, r1
        B exit
neg     SUB r1, r1, r0
exit    ...         ; program continues
```

Branch penalty if N < M



Conditional execution

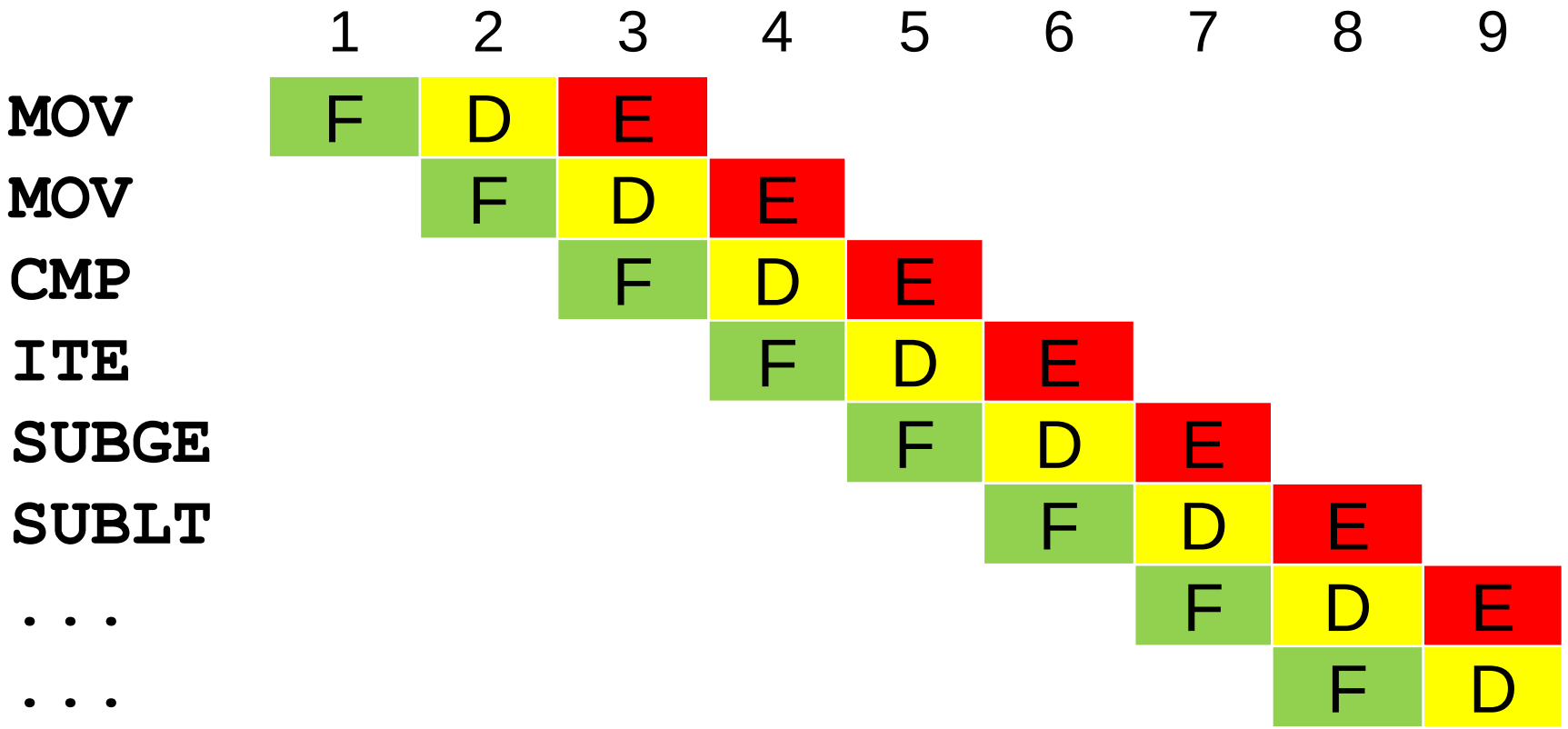
- The IT (If-Then) block avoids branch penalty because there is no change to program flow:

```
ITxyz <cond>
instr1<cond> <operands>
instr2<cond OR not cond> <operands>
instr3<cond OR not cond> <operands>
instr4<cond OR not cond> <operands>
```

Absolute value of N – M with IT

```
        MOV r0, #N
        MOV r1, #M
        CMP r0, r1
        ITE GE
        SUBGE r0, r0, r1
        SUBLT r1, r1, r0
exit    ...         ; program continues
```

No branch penalty



IT syntax

- first statement after `IT` must be the true case
- up to 4 instructions (true or false) are allowed
- the number of instructions in true and false cases must match the number of `T` and `E`
- false condition is inverse of true condition
- branches to `IT` instructions are not allowed
- an `IT` instruction can be a branch only if it is the last one.