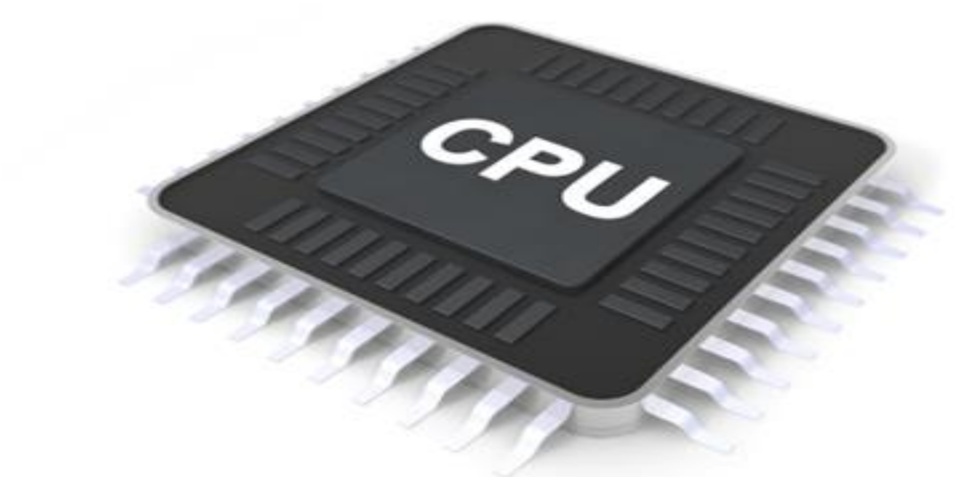

# Jumps and Branches

ENEE 3582

Microp

# Jumps

- Unconditional jump
- Changes the address in PC
- JMP: jump to address
    - Format: `JMP k          ;PC = k`
    - k is unsigned 22 bits: 0 to <4M
- RJMP: Relative jump
    - Format: `RJMP k         ;PC = PC + k + 1`
    - k is signed 11 bits: -2K ≤ k < 2K
- IJMP: indirect jump
    - Format: `IJMP           ;PC = mem[Z]`
    - Mem[Z] is a word

# Conditional Branches 1

- Branch if a condition is True
- Condition is based on 1 flag
- k is signed 7 bit: -64 to 63
- Relational branches:
  - If condition is TRUE: PC = PC + k + 1
  - If condition is FALSE: PC = PC + 1

| Branch | Operand | Description | Condition |
|---|---|---|---|
| BRCC | k | Branch if No Carry | C==0 |
| BRCS | k | Branch if Carry | C==1 |
| BRMI | k | Branch if Negative | N==1 |
| BRPL | k | Branch if Positive | N==0 |
| BRNV | k | Branch if No Overflow | V==0 |
| BRVS | k | Branch if Overflow | V==1 |

# Conditional Branching & Comparison

- First a comparison is made between two operands: Op1, Op2
- The comparison is a form a subtraction
- Flags are set
- Branch based on condition of flag(s)

# Comparison of Signed Operands

- Op1, Op2: Signed
- Op1 = Op2:
  - Equal
  - Flags: $Z = 1$
- Op1 < Op2:
  - Less than
  - Flags: $N \oplus V = 1$
- Op1 > Op2:
  - Greater than = Not equal and not less than
  - Flags: $(Z = 0)$ and $(N \oplus V = 0)$

# Conditional Branches: Signed Comparison

- First a comparison is made between 2 values
- Signed comparison: greater than, less than, equal
- k is signed 7 bit: -64 to 63
- Relational branches:
  - If condition is TRUE: PC = PC + k + 1
  - If condition is FALSE: PC = PC + 1

| **Branch** | **Operand** | **Description** | **Condition** | **Flags** |
|---|---|---|---|---|
| `BREQ` | k | Branch if equal to | Op1 == Op2 | Z == 1 |
| `BRNE` | k | Branch if not equal to | Op1 ≠ Op2 | Z == 0 |
| `BRLT` | k | Branch if less than | Op1 < Op2 | N ⊕ V == 1 |
| `BRGE` | k | Branch if greater than or equal to | Op1 ≥ Op2 | (Z == 1) OR (N ⊕ V == 0) |

# Comparison of Unsigned Operands

- Op1, Op2: Unsigned
- Op1 = Op2:
  - Equal/Same
  - Flags: Z = 1
- Op1 < Op2:
  - Lower
  - Flags: C = 1
- Op1 > Op2:
  - Higher = not the same and not lower
  - Flags: (Z = 0) and (C = 0)

# Conditional Branches: Unsigned Comparison

- First a comparison is made between 2 values
- Signed comparison: higher, lower, same/equal
- k is signed 7 bit: -64 to 63
- Relational branches:
  - If condition is TRUE: PC = PC + k + 1
  - If condition is FALSE: PC = PC + 1

| Branch | Operand | Description | Condition | Flags |
|---|---|---|---|---|
| BREQ | k | Branch if equal to | Op1 == Op2 | Z == 1 |
| BRNE | k | Branch if not equal to | Op1 ≠ Op2 | Z == 0 |
| BRLO | k | Branch if lower | Op1 < Op2 | C == 1 |
| BRSH | k | Branch if same or higher | Op1 ≥ Op2 | (Z== 1) OR (C == 0) |

# Equality and Inequality Logic

- Equality Logic:
  - EQ, NE: ==, !=
  - Negative:
    - NOT (EQ) = NE
    - NOT (NE) = EQ
- Inequality Logic:
  - GT, GE: >, ≥
  - LT, LE : <, ≤
  - Negative:
    - NOT(GT) = LTE NOT(>) = ≤
    - NOT(GE) = LT NOT(≥) = <
    - NOT(LT) = GE NOT(<) = ≥
    - NOT(LE) = GT NOT(≤) = >

# Coding BRGT, BRLE

- BRLE doesn’t exist
  - Can be codes using 2 consecutive branches
  - E.g. Code as: (LT) OR (EQ)

```
          BRLT  dest
          BREQ  dest
```

- BRGT doesn’t exist
  - Can be codes using 2 consecutive branches
  - E.g. Code as: (NOT(EQ)) AND (GE))

```
          BREQ  skip
          BRGE  dest
     skip:
```

# Comparison

- `CP:` Compares 2 regs
  - Format: `CP Rm, Rn` `;SREG <- (Rm-Rn)`
  - Rm,Rn can be R0…R31
- `CPC:` Compares 2 regs with carry
  - Format: `CPC Rm, Rn` `;SREG <- (Rm-Rn-C)`
  - Rm,Rn can be R0…R31
- `CPI:` Compares a reg to an immediate
  - Format: `CPI Rm, k` `;SREG <- (Rm-k)`
  - k is unsigned 8bits: 0…255
  - Rm can be R16…R31
- `TST:` Compares a reg to 0
  - Format: `TST Rm` `;SREG <- (Rm–0)`
  - Rm can be R16…R31

# if-then Structure

```
if (condition)
   then …                ;execute if condition is TRUE
endif
```

- Steps for single condition IF-THEN:
  1. Load values in Rm, Rn/k
  2. Compare
  3. Branch if condition is FALSE (ie negative condition) to ENDIF
  4. Execute next instruction if condition is TRUE (THEN clause)

# if-then Single Condition ==

```
if val1 == val2
      then …

endif
```

```
      LDI Zh, HIGH(2*val1)
      LDI Zl, LOW(2*val1)
      LPM R16, Z
      LDI Zh, HIGH(2*val2)
      LDI Zl, LOW(2*val2)
      LPM R17, Z

      CP R16, R17
      BRNE endif
then: …

endif:
```

# if-then Single Condition !=

```
if val1 != val2
        then …

endif
```

```
        LDI Zh, HIGH(2*val1)
        LDI Zl, LOW(2*val1)
        LPM R16, Z
        LDI Zh, HIGH(2*val2)
        LDI Zl, LOW(2*val2)
        LPM R17, Z

        CP R16, R17
        BREQ endif
then:   …

endif:
```

# Compare and Skip if Equal

- Format: `CPSE Rm, Rn ;SREG <- (Rm-Rn)`
- If Rm == Rn
  - then: PC = PC + 2
  - else PC = PC + 1
- Rm,Rn can be R0…R31
- Example:

```
if val1 != val2
        then …
endif
```

```
          CPSE R16, R17
then: …
endif:
```

# if-then Single Condition <

```
if val1 < val2
        then …

endif
```

```
        LDI Zh, HIGH(2*val1)
        LDI Zl, LOW(2*val1)
        LPM R16, Z
        LDI Zh, HIGH(2*val2)
        LDI Zl, LOW(2*val2)
        LPM R17, Z

        CP R16, R17
        BRGE endif
then:   …

endif:
```

# if-then Single Condition >

```
if val1 > val2

      then …

endif
```

```
        LDI Zh, HIGH(2*val1)
        LDI Zl, LOW(2*val1)
        LPM R16, Z
        LDI Zh, HIGH(2*val2)
        LDI Zl, LOW(2*val2)
        LPM R17, Z

        CP R16, R17
        BRLT endif
        BREQ endif
then:   …

endif:
```

# if-then Single Condition <=

```
if val1 <= val2

       then …

endif
```

```
      LDI Zh, HIGH(2*val1)
      LDI Zl, LOW(2*val1)
      LPM R16, Z
      LDI Zh, HIGH(2*val2)
      LDI Zl, LOW(2*val2)
      LPM R17, Z

      CP R16, R17
      BREQ then
      BRGE endif
then: …

endif:
```

# if-then Single Condition >=

```
if val1 <= val2
        then …

endif
```

```
        LDI Zh, HIGH(2*val1)
        LDI Zl, LOW(2*val1)
        LPM R16, Z
        LDI Zh, HIGH(2*val2)
        LDI Zl, LOW(2*val2)
        LPM R17, Z

        CP R16, R17
        BRLT endif
then:   …

endif:
```

# if-then 2 Conditions Case 1

- Case 1: (condition 1) AND (condition 2)

```
if (condition 1) && (condition 2)
          then …
endif
```

- 2 conditions must be TRUE to execute THEN
- Negative: NOT(condition 1) OR NOT(condition 2)
- 1 condition must be false to ENDIF
- Steps for 2 condition IF-THEN case1:
  1. Load values for comparisons
  2. Branch to ENDIF if condition1 is FALSE
  3. Branch to ENDIF if condition2 is FALSE
  4. Execute THEN clause

# if-then Case 1 Example

```
                                           ;R16 = val1
                                           ;R17 = val2
                                           ;R18 = val3
                                           ;R19 = val4

if (val1 == val2) && (val3 < val4)         CP R16, R17        ;val1 ? val2
                                           BRNE endif
                                           CP R18, R19        ;val3 ? val4
                                           BRGE endif
   then …                           then:  …

endif                               endif:
```

# `if-then` 2 Conditions Case 2

- Case 2: (condition 1) OR (condition 2)

```
if (condition 1) || (condition 2)
     then …
endif
```

- 1 conditions must be TRUE to execute then
- Negative: NOT(condition 1) AND NOT(condition 2)
- 2 condition must be false to endif
- Steps for 2 condition IF-THEN case2:
  1. Load values for comparisons
  2. Branch to THEN clause if condition1 is TRUE
  3. Branch to ENDIF if condition2 is FALSE
  4. Execute THEN clause

# if-then Case 2 Example

```
                                       ;R16 = val1
                                       ;R17 = val2
                                       ;R18 = val3
                                       ;R19 = val4

if (val1 == val2) || (val3 < val4)     CP    R16, R17
                                       BREQ  then
                                       CP    R18, R19
                                       BRGE  endif

   then …                        then: …

endif                            endif:
```

# if-then-else Structure

```
if (condition)
    then …                ;execute if condition is TRUE
    else …                ;execute if condition is FALSE
endif
```

- Steps for single condition IF-THEN-ELSE:
  1. Load values in Rm, Rn/k
  2. Compare
  3. Branch if condition is FALSE to ELSE clause
  4. Execute THEN clause otherwise
  5. Jump at the end of THEN clause to ENDIF

# if/then/else Single Condition TRUE

```
if val1 == val2
        then
                …
        else
                …
endif
```

```
        LDI Zh, HIGH(2*val1)
        LDI Zl, LOW(2*val1)
        LPM R16, Z
        LDI Zh, HIGH(2*val2)
        LDI Zl, LOW(2*val2)
        LPM R17, Z

        CP R16, R17
        BRNE else
then:   …
        RJMP endif
else:   …

endif:
```

# repeat-until aka Looping

```
repeat:
    …
until (condition)    ;stop if condition is TRUE, branch if FALSE
```

- Steps for single condition REPEAT-ELSE:
  1. Load values in Rm, Rn/k
  2. Create a top label
  3. Execute the body of the structure
  4. Branch if condition is FALSE to top label

# repeat-Until Single Condition

```
repeat:
      …

until (val1 == val2)
```

```
                LDI Zh, HIGH(2*val1)
                LDI Zl, LOW(2*val1)
                LPM R16, Z
                LDI Zh, HIGH(2*val2)
                LDI Zl, LOW(2*val2)
                LPM R17, Z


repeat:             …


                CP R16, R17
                BRNE repeat
```

# while Structure

```
while (condition)    ;Repeat structure if TRUE. Stop if False
    …
endw
```

- Steps for single condition WHILE:
  1. Load values in Rm, Rn/k
  2. Compare
  3. If condition is FALSE branch to ENDW
  4. Execute the body of the structure
  5. JMP to step 2

# while Single Condition

```
           LDI Zh, HIGH(2*val1)
           LDI Zl, LOW(2*val1)
           LPM R16, Z
           LDI Zh, HIGH(2*val2)
           LDI Zl, LOW(2*val2)
           LPM R17, Z
```

```
while (val1 == val2)
      …

endw
```

```
while: CP R16, R17
       BRNE endw
       …
       JMP while
endw:
```