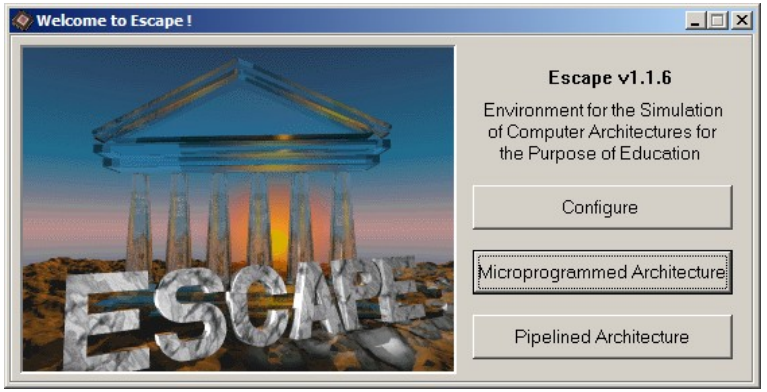
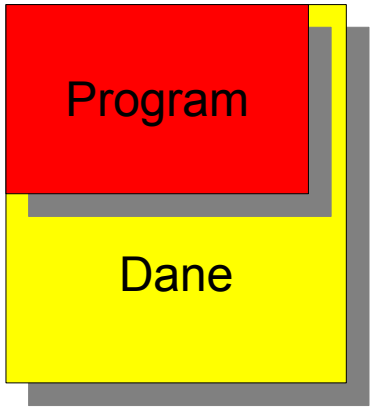
Symulator Escape

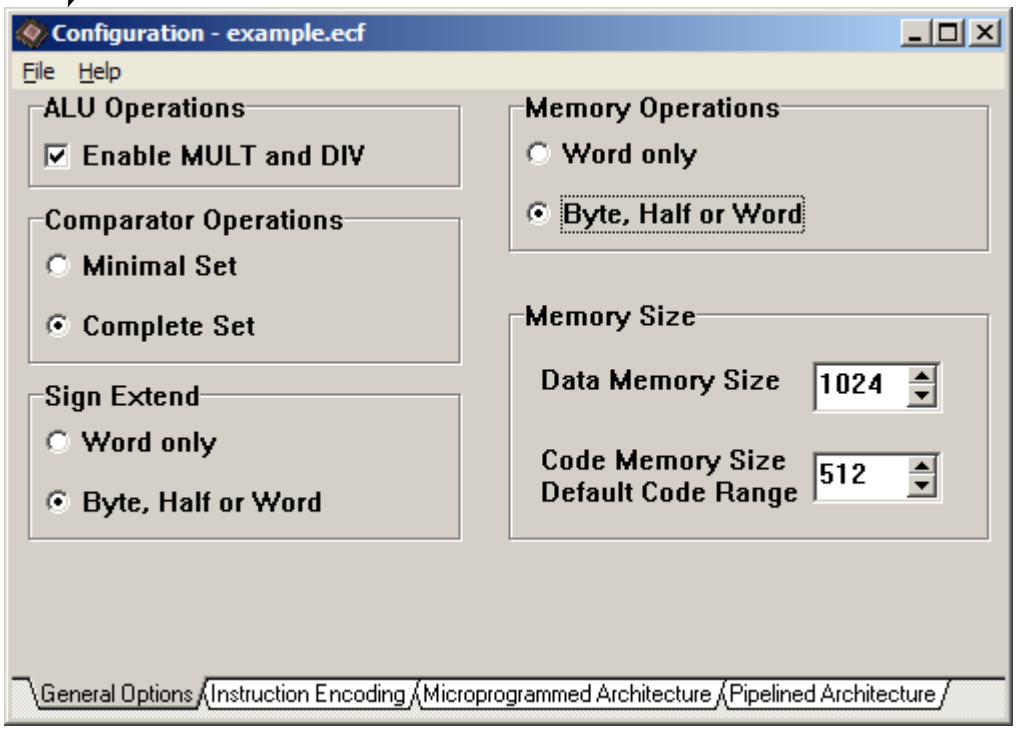
Konfiguracja ogólna



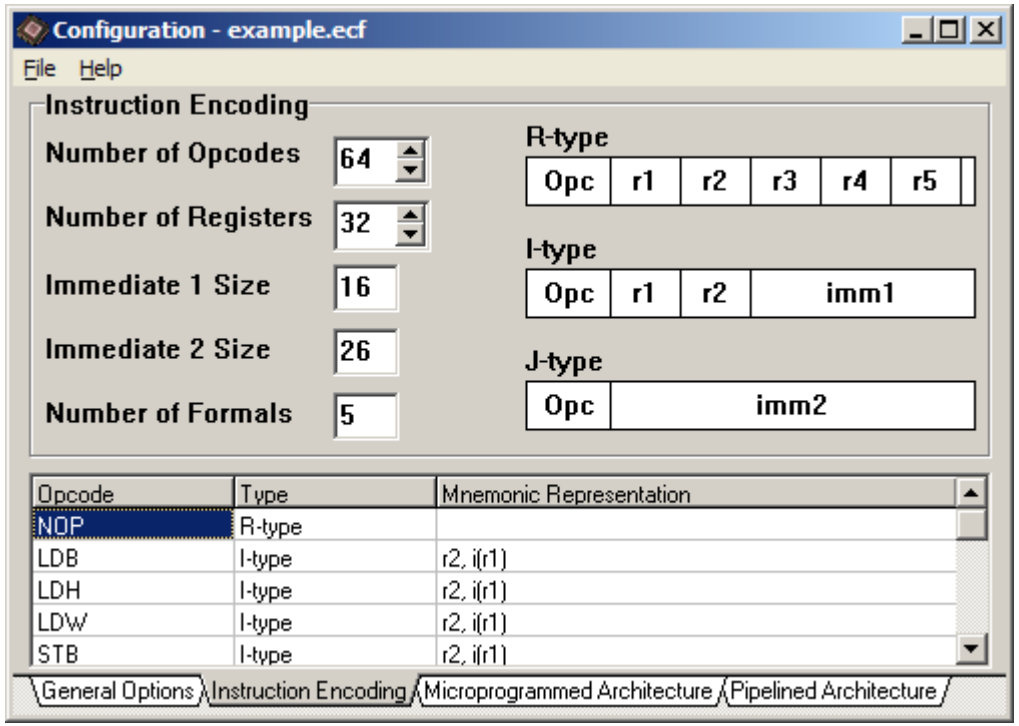
Załaduj konfigurację symulatora
(*File -> OpenFile*)
z pliku **`example.ecf`**

Enable MUL and DIV
Complete Set of Comp.Oper
Sign Extension of B/H/W
Memory Oper on B/H/W



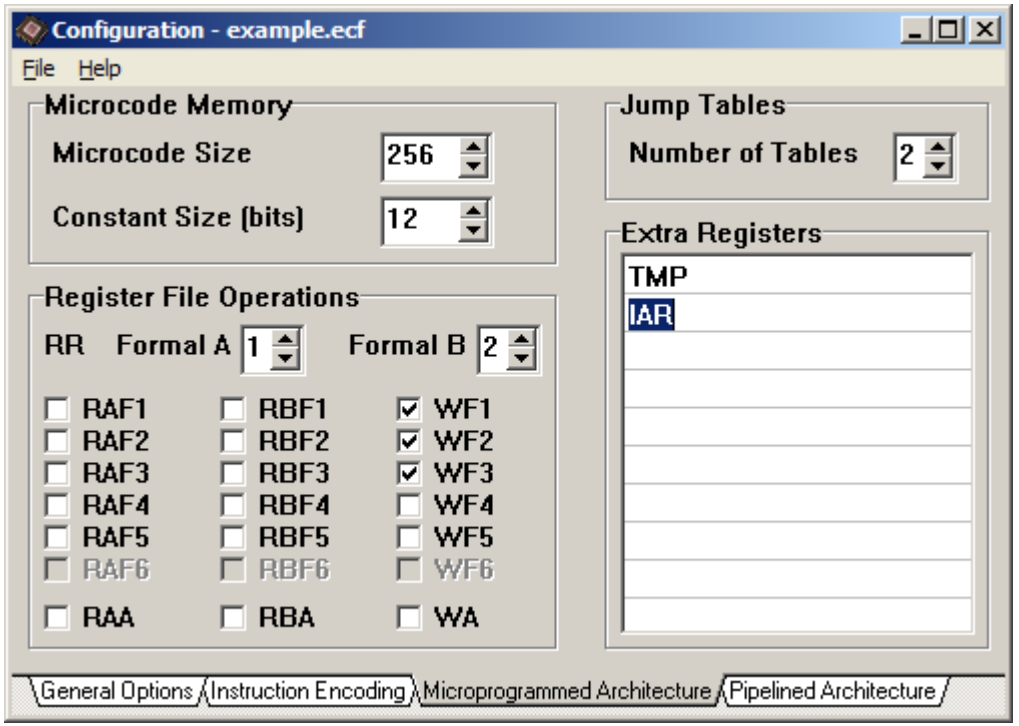


Lista i format instrukcji



64 typy instr. → 6 bitowe pole typu instrukcji (*opcode*)
32 rejestry wewn. → 5 bitowe numery rejestrów (*formal fields: r1,r2,...*), R0 ... R31
Stałe w kodzie instrukcji (*immediate*):
imm1 – 16 bitów dla instrukcji Load/Store oraz skoków warunkowych (Bxx)
imm2 – 26 bitów dla instrukcji skoku bezwarunkowego (JMP)

Obsługa pliku rejestrów i mikrokod



Pamięć mikrokodu → 256 12-bitowych słów

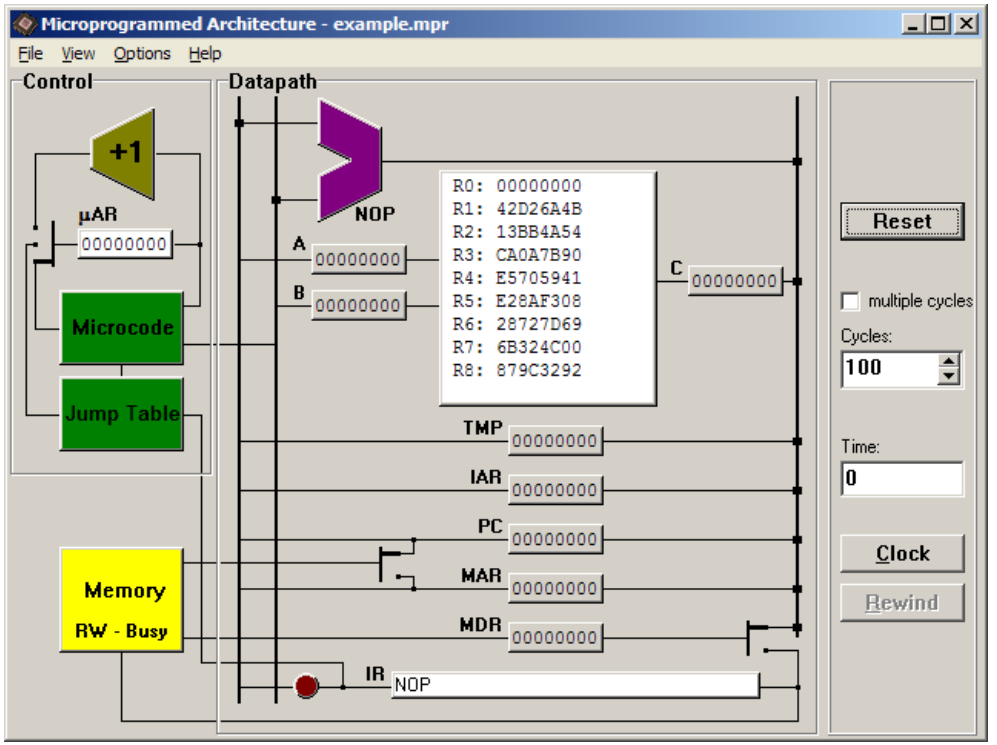
Operacje na pliku rejestrów:
odczyt: *RR* (*Read Registers*) → odczyt do A i B zaw. rej. o numerze z pól *r1* i *r2*
zapis: *WF1*, *WF2*, *WF3* (*Write Formal n*) → zapis do rejestru o numerze z pola *r1*, *r2*, *r3*

Tablice skoków dla sterowania mikrokodem → 2

Rejestry dodatkowe (niezależnie od rejestrów R0..R31)

Projekt

Załaduj projekt
(*File -> OpenFile*)
z pliku **`example.mpr`**



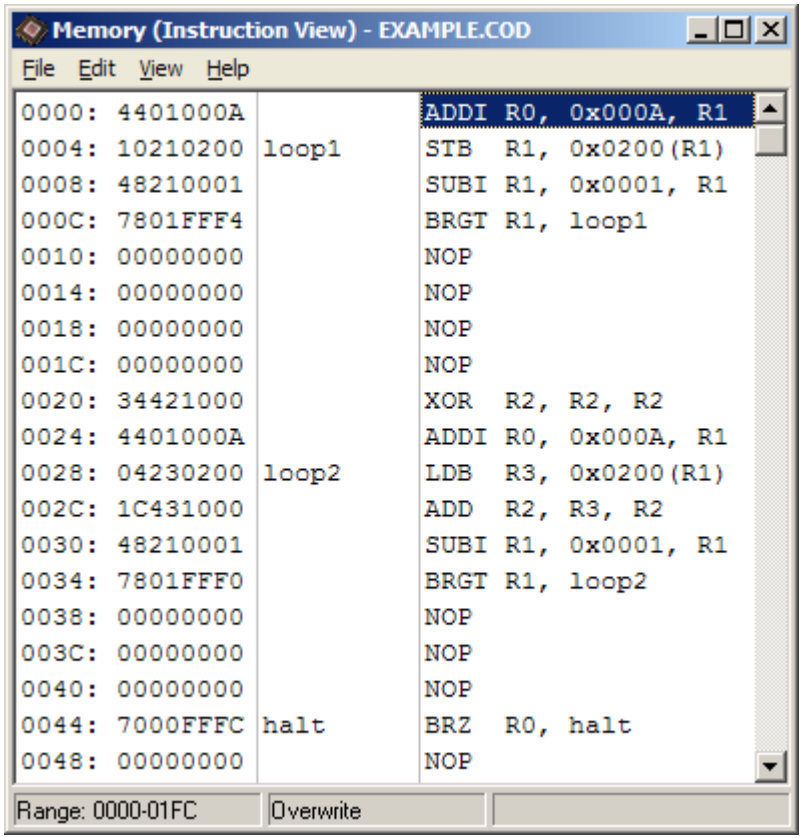
Projekt
(.mpr – microcode project)

Program w asemblerze
(.cod - code)

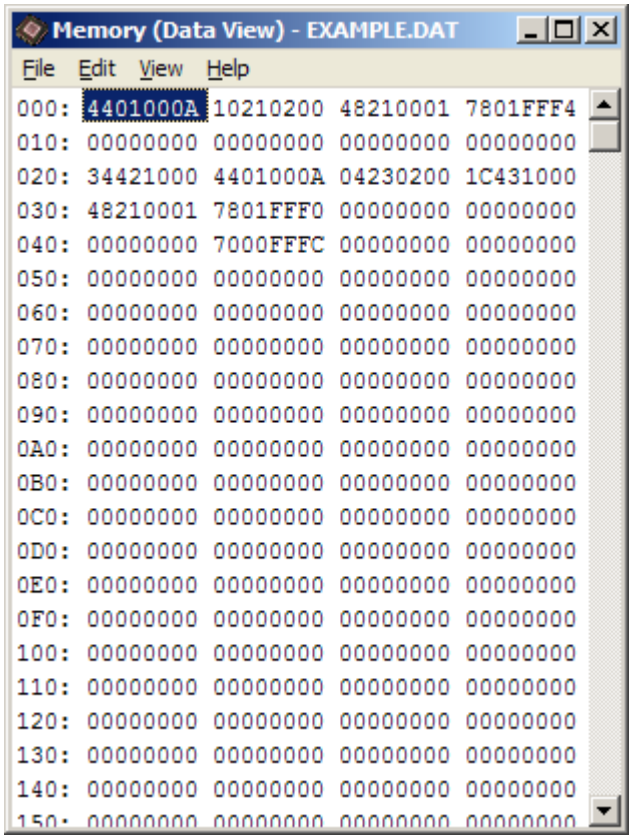
Mikrokod
(.mco - microcode)

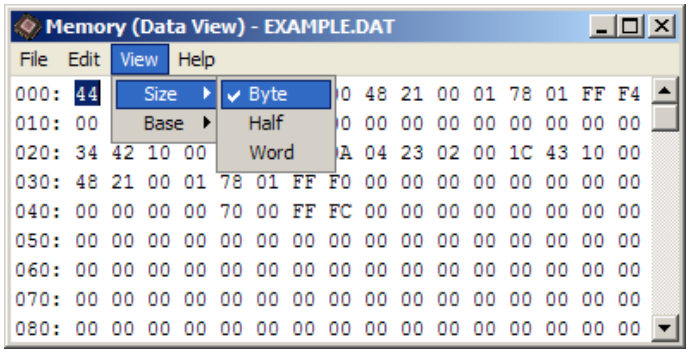
Zawartość pamięci
(.dat - data)

Podgląd pamięci

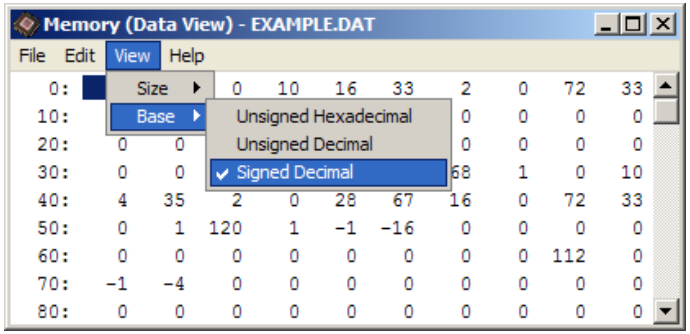


podgląd kodu maszynowego

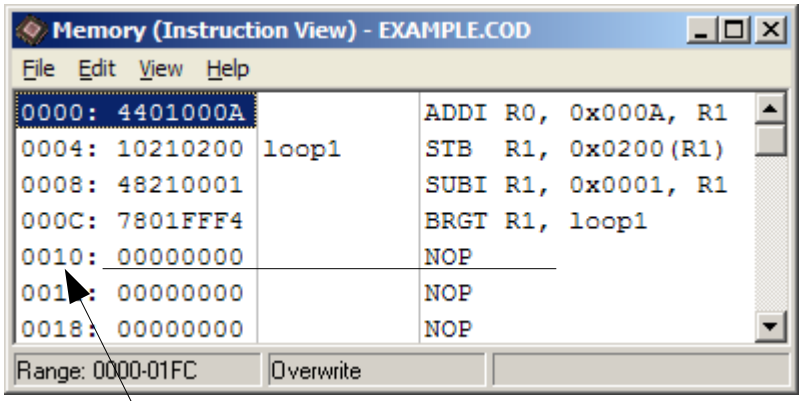


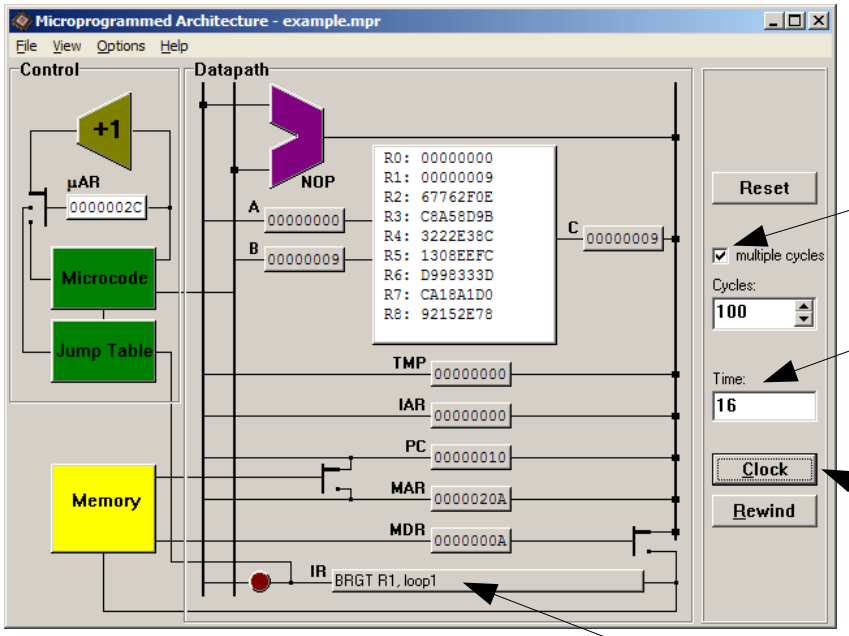


Mozliwość wybór rozmiaru(B/H/W) oraz kodowania (hex, un-, signed)



Praca krokowa i pułapki





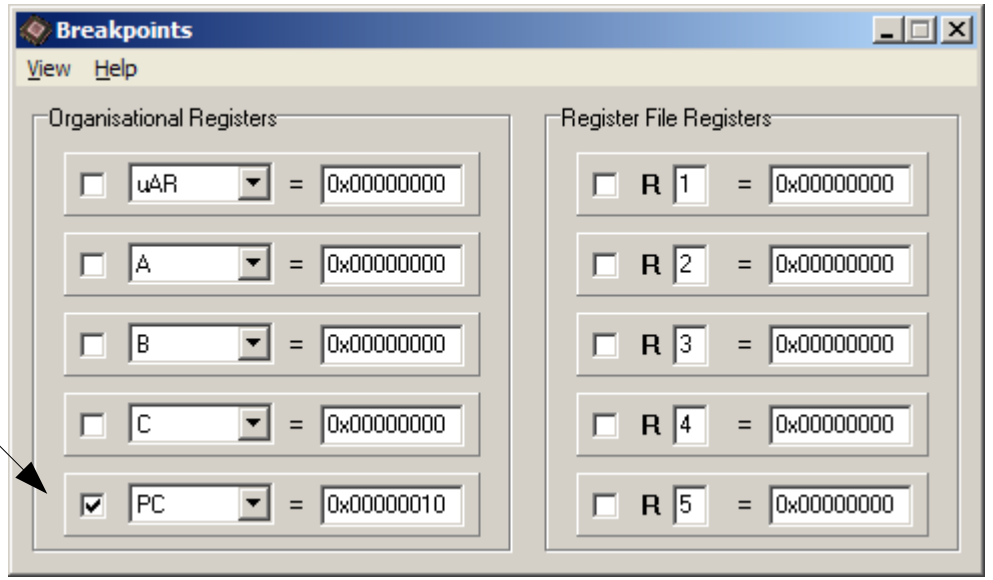
Zezwolenie na wykonanie wielu cykli zegara na raz

Czas (liczba cykli) działania programu

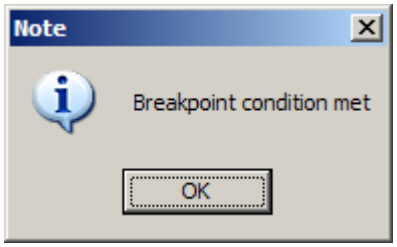
Wymuszenie 1 lub wielu cykli zegara

Aktualnie wykonywana (pobrana) instrukcja

ustawienie pułapki na wartość 10 dla PC



! Uwaga: pułapka na np. PC=10 powoduje zatrzymanie programu gdy w PC pojawi się wartość 10, ale nie oznacza to, że instrukcja spod adresu 10 została wykonana



Instrukcje transferu

```
LDB   Ry,i(Rx)     Load Byte:   Ry.B ← Mem[Rx+i]
LDH   Ry,i(Rx)     Load Half:   Ry.H ← Mem[Rx+i]
LDW   Ry,i(Rx)     Load Word:   Ry.W ← Mem[Rx+i]

STB   Ry,i(Rx)     Store Byte:  Mem[Rx+i] ← Ry.B
STH   Ry,i(Rx)     Store Byte:  Mem[Rx+i] ← Ry.H
STW   Ry,i(Rx)     Store Byte:  Mem[Rx+i] ← Ry.W
```

Format instrukcji LD i ST

Rx,Ry,Rz – rejestry wewnętrzne procesora: R0...R31
i – stała (liczba) 16-bitowa (2B), podczas operacji rozszerzana znakowo do 32-bitów
label – etykieta linii programu w asemblerze (adres skoku)
.B – bajt (8-bitów), Rx.B - najmniej znaczący bajt rejestru
.H – półsłowo (16-bitów), Rx.H - mniej znaczące półsłowo rejestru
.W – słowo (32-bity), Rx.W – cała zawartość 32-bitowego rejestru

Instrukcje arytmetyczne i logiczne

```
ADD   Rx,Ry,Rz     Add:   Rz ← Rx+Ry
SUB   Rx,Ry,Rz     Sub:   Rz ← Rx-Ry
MUL   Rx,Ry,Rz     Mul:   Rz ← Rx*Ry
DIV   Rx,Ry,Rz     Div:   Rz ← Rx/Ry

ADDI  Rx,i,Ry      Add:   Ry ← Rx+i
SUBI  Rx,i,Ry      Add:   Ry ← Rx-i
MULI  Rx,i,Ry      Add:   Ry ← Rx*i
DIVI  Rx,i,Ry      Add:   Ry ← Rx/i

AND   Rx,Ry,Rz     And:   Rz ← Rx&Ry
OR    Rx,Ry,Rz     Or :   Rz ← Rx|Ry
XOR   Rx,Ry,Rz     Xor:   Rz ← Rx^Ry

ANDI  Rx,i,Ry      And:   Ry ← Rx&i
ORI   Rx,i,Ry      Or :   Ry ← Rx|i
XORI  Rx,i,Ry      Xor:   Ry ← Rx^i

SLL   Rx,Ry,Rz     Shift Left  Logical:   Rz ← Rx<<Ry
SRL   Rx,Ry,Rz     Shift Right Logical:   Rz ← Rx>>Ry
SRA   Rx,Ry,Rz     Shift Right Arithm.:   Rz ← Rx>>Ry with sign
SLLI  Rx,i,Ry      Shift Left  Logical:   Ry ← Rx<<i
SRLI  Rx,i,Ry      Shift Right Logical:   Ry ← Rx>>i
SRAI  Rx,i,Ry      Shift Right Arithm.:   Ry ← Rx>>i with sign
```

Instrukcje skoków i pozostałe

Dla wszystkich skoków warunkowych:

```
BRcc: PC ← label(PC+offset) if condition(cc)==True

BRZ  Rx,label   Branch if Zero             : condition Rx = 0
BRNZ Rx,label   Branch if Not Zero         : condition Rx ≠ 0
BRGT Rx,label   Branch if Greater Than     : condition Rx > 0
BRGE Rx,label   Branch if Grater or Equal  : condition Rx ≥ 0
BRLT Rx,label   Branch if Less Than        : condition Rx < 0
BRLE Rx,label   Branch if Less of Equal    : condition Rx ≤ 0
```

Inne:

```
NOP          No Operation
LIH Rx,i     Load Immediate High: Rx[31..16] ← i.H
```

Tryby adresowania

Tryby adresowania są sposobem na zapisywanie lokalizacji operandów, biorących udział w operacji

Adresowanie natychmiastowe (*Immediate*) – operand w kodzie instrukcji

```
ADDI R5,0x55AA,R3
```

Adresowanie rejestrowe bezpośrednie (*Register Direct*) – operand w rejestrze

```
LDW R7,0x20(R3)
```

Adresowanie rejestrowe pośrednie z przesunięciem (*Register Indirect with Displacement*)
- operand w komórce pamięci, której adres jest w rejestrze (+ przesunięcie)

Assembler – przykład: n!

R1 – wejściowa wartość n
R2 – wynik

! Transfery pomiędzy rejestrami realizowane są za pomocą dodawania R0 (który ma zawsze wartość zero)

Metoda iteracyjna: n! = n*(n-1)*(n-2)*...*1

```
        ADDI  R0,0x000A,R1     ładowanie n=10 do R1
        ADD   R1,R0,R2         ładowanie do R2 wartość początkowej (n)
next    SUBI  R1,0x0001,R1     dekrementacja n (R1 ← R1-1)
        BRZ   R1,halt          skok gdy n=0 (koniec obliczeń)
        MUL   R2,R1,R2         mnożenie wyniku przez n-1 (R2 ← R2*R1)
        BRZ   R0,next          skok bezwarunkowy (następny cykl obliczeń)
halt    BRZ   R0,halt          stop
```

etykieta

mnemonik

operandy (zapisane za pomocą różnych trybów adresowania)