MIPS64

INTRODUCTION

Ernesto Sanchez
ernesto.sanchez@polito.it

CAD GROUP
www.cad.polito.it

Outline

- MIPS64: Introduction
- Assembler programs: How to Write
- WinMIPS64 the initial glance.

MIPS64

- Generalities
  - MIPS (**M**icroprocessor without **I**nterlocked **P**ipeline **S**tages) is a family of RISC processors, which have been very successful for embedded applications
  - The first processor in the family was introduced in 1985
  - Several versions have been introduced since then.

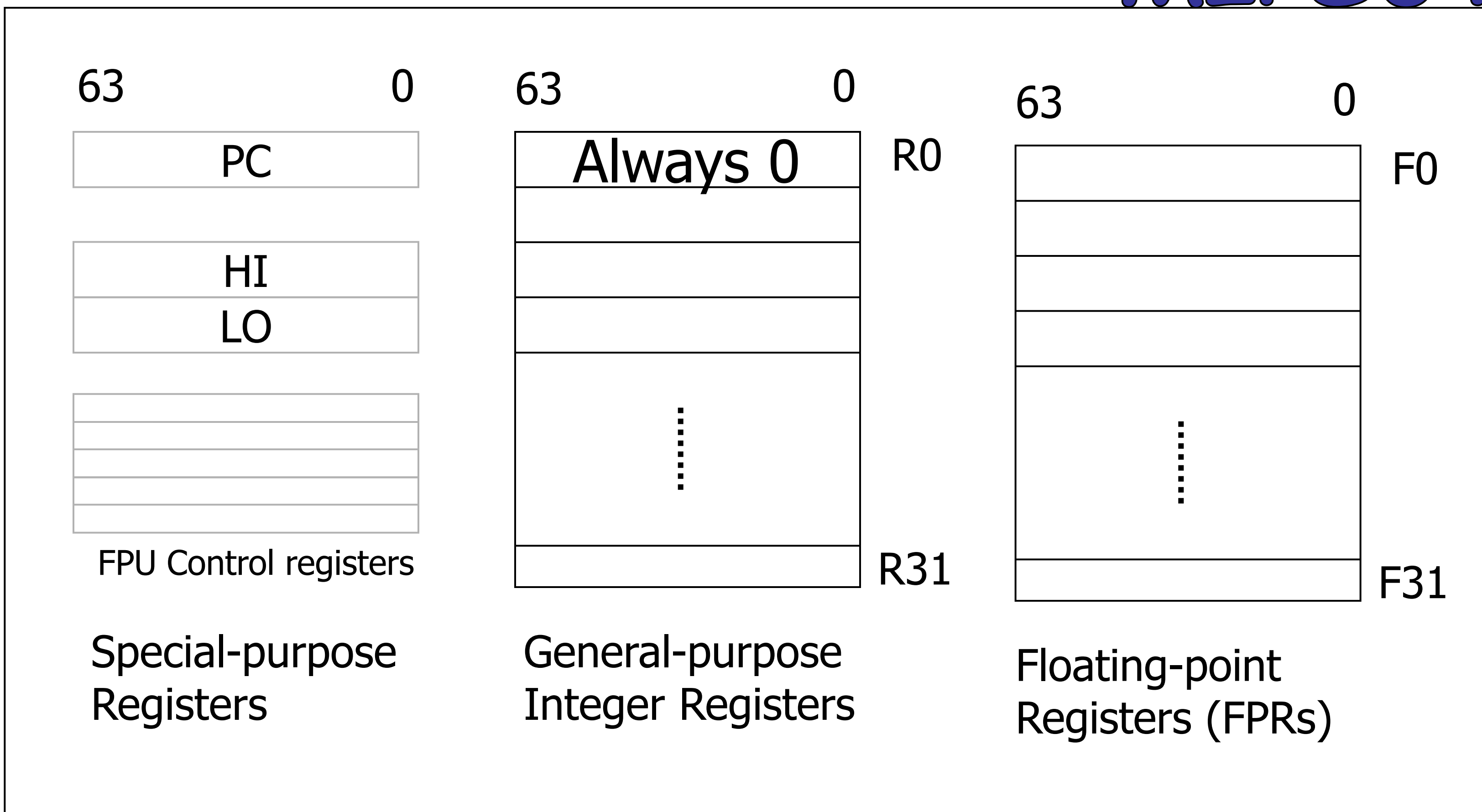
MIPS64

- Generalities
  - Simple load-store Instruction Set
  - Designed for pipeline efficiency
    - Fixed instruction length
    - Low-power applications

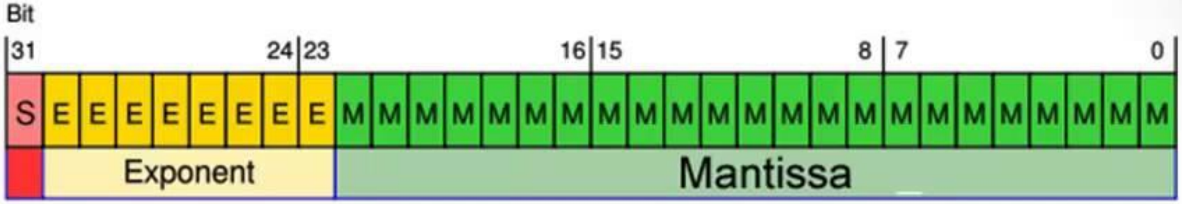
  - What is described here is a simplified version of the so called MIPS64.

MIPS64 – Programmer's Model

MIPS64



IEEE 754 32 bit -> binario



31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

$$(-1)^{0} * 10^{10000110-01111111} * 1{,}010000011$$

$$1 \quad * 10^{111} * 1{,}010000011$$

$$10100000{,}11$$

160,75

Data Types

- Byte (8 bits)
- Half Words (16 bits)
- Words (32 bits)
- Double Words (64 bits)
- 32-bit single precision floating-point
- 64-bit double precision floating-point.

Addressing Modes

- It uses 16 bits Immediate field
  - DADDUI R1, R2, #32

    $R1 \leftarrow R2 + 32$

  - DADDUI R1, R0, #32

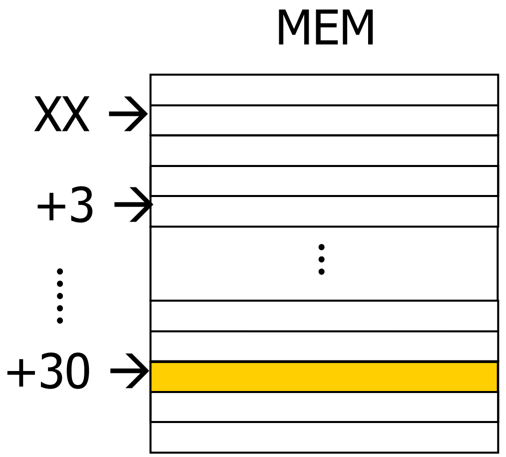
    $R1 \leftarrow 32$

Addressing Modes

- Displacement
  - LD R1, 30(R2)

R2 = XX

R1 ← MEM[R2 + 30]

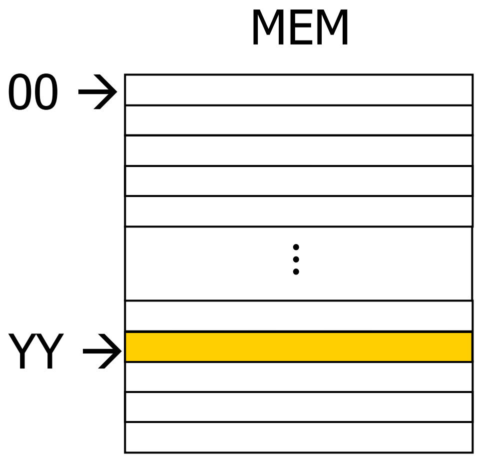


Addressing Modes

- Displacement
  - LD R1, 0(R2) → *Register Indirect*

$R2 = YY$

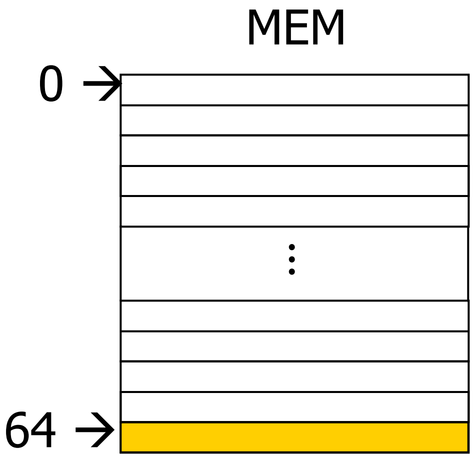
$R1 \leftarrow MEM[R2]$



Addressing Modes

- Displacement
  - LD R1, 64(R0) → *Absolute Addressing*

R1 ← MEM[64]



Instruction Format

- A CPU instruction is a single 32-bit aligned word
  - Include a 6-bit primary opcode

31	0
INSTRUCTION	

- The CPU instruction formats are:
  - Immediate
  - Register
  - Jump.

Instruction Format – Immediate

- I – type instruction

6	5	5	16
OPcode	Rs	Rt	Immediate

Field	**Description**
opcode	6-bit primary operation code
Rs	5-bit specifier for the source register
Rt	5-bit specifier for the target (source/destination) register
Immediate	16-bit signed *immediate* used for logical operands, arithmetic signed operands, load/store address byte offsets, and PC-relative branch signed instruction displacement

Instruction Format – Register

- R – type instruction

6	5	5	5	5	6
OPcode	Rs	Rt	Rd	Sa	Function

Field	**Description**
opcode	6-bit primary operation code
Rd	5-bit specifier for the destination register
Rs	5-bit specifier for the source register
Rt	5-bit specifier for the target (source/destination) register
Sa	5-bit shift amount
Function	6-bit function field used to specify functions within the primary opcode SPECIAL

Instruction Format – Jump

- J – type instruction

6	26
OPcode	Offset added to PC

Field	Description
opcode	6-bit primary operation code
Offset	26-bit index shifted left two bits to supply the low-order 28 bits of the jump target address

INSTRUCTION SET

- Grouped By Function
  - Load and store
  - ALU operations
  - Branches and Jumps
  - Floating Point
  - Miscellaneous

Each instruction is 32 bits long.

Load and Store

- MIPS processors use a load/store architecture
- Main memory is accessed only through load and store instructions.

Load and Store – Examples

- LD load double word

```
LD R1, 28(R8)            ;R1 ← MEM[R8 + 28]
```

- LB load Byte

LB R1, 28(R8) ;R1 $\leftarrow$ $(\text{[MEM[R8 + 28]]}_7)^{56}$ ## MEM[R8 + 28]

- LBU load Byte unsigned

LBU R1, 28(R8) ;R1 $\leftarrow$ 0^{56} ## MEM[R8 + 28]

Load and Store – Examples

- L.S load FP Single

```
L.S F4, 46(R5)     ;F4 ← MEM[R5 + 46] ## 0^32
```

- L.D load FP Double

```
L.D F4, 46(R5)     ;F4 ← MEM[R5 + 46]
```

- SD Store Double

```
SD R1, 28(R8)      ;MEM[R8 + 28]← R1
```

Load and Store – Examples

- SW Store Word

```
SW R1, 28(R8)          ;MEM[R8 + 28]←32 R1 LSB
```

$\text{SW R1, 28(R8)} \quad ;\text{MEM[R8 + 28]} \leftarrow_{32} \text{R1 } \textit{LSB}$

- SH Store Half Word

$\text{SH R1, 28(R8)} \quad ;\text{MEM[R8 + 28]} \leftarrow_{16} \text{R1 } \textit{LSB}$

- SB Store byte

$\text{SB R1, 28(R8)} \quad ;\text{MEM[R8 + 28]} \leftarrow_{8} \text{R1 } \textit{LSB}$

Load and Store – Examples

- S.S Store FP Single

```
S.S F4, 28(R8)                ;MEM[R8 + 28]←32 F4 63..32
```

- S.D Store FP Double

```
S.D F4, 28(R8)                ;MEM[R8 + 28]← F4
```

ALU operations

- All operations are performed on operands held in processor registers
- Instruction types
  - Immediate and three-operand Instructions
  - Two-operand Instructions
  - Shift instructions
  - Multiply and divide instructions
- 2’s complement arithmetic
  - Add
  - Subtract
  - Multiply
  - Divide.

ALU operations: R0 usage

- ADD immediate with R0 as source operand
  - **loading a constant**

```
DADDUI R1,R0,25        ;R1 ← 25
```

- ADD Rx with R0 as source operand
  - **register to register**.

```
DADD R1,R0,R2          ;R1 ← R2
```

ALU – Examples

- DADDU Double Add unsigned

```
DADDU R1,R2,R3        ;R1 ← R2 + R3
```

- DADDUI Double Add Unsigned Immediate

```
DADDUI R1,R2,74       ;R1 ← R2 + 74
```

- LUI Load Upper Immediate

LUI R1,0x47 ;R1 ← $0^{63..32}$ ## 0x47 ## $0^{15..0}$

```
DADDUI R1,R1,0x13     ;R1 ← R1 + 0x13
                      ;R1 ← 0x470013
```

HowTo: 32-bit constant values

- LUI + ORI:

; +2,147,483,647 -> 0x7FFF_FFFF

`lui r7, 0x7FFF` ; r7 = 0000_ 0000_7FFF_ 0000

`ori r7, r7, 0xFFFF ;`

; From now

;r7 = 0000_ 0000_7FFF_ FFFF

HowTo: 32-bit constant values

0000_ 0000_7FFF_0000

0000_ 0000_0000_FFFF

0000_ 0000_7FFF_ FFFF

NOTE: ALL Logical instructions extend the immediate with 0

A	B	A or B
0	0	0
0	1	1
1	0	1
1	1	1

ALU – Examples

- DSLL Double Shift left logical

```
DSLL R1,R2,3        ;R1 ← R2 <<3
```

- SLT Set Less than

```
SLT R1,R2,R3        ;IF (R2 < R3) R1 ← 1
                    ;ELSE R1 ← 0
```

HowTo: 32-bit constant values

NOTE: 0xC = 0b1100

- How to obtain:

r7 = 0000 _ 0000 _C1A0_FEDE

- LUI + ORI:

FFFF_FFFF_C1A0_FEDE

- Shift Left Logical 32 bits:

C1A0_FEDE_0000_0000

- Shift Right Logical 32 bits:

0000 _ 0000 _C1A0_FEDE

Branch and Jump

- PC-relative conditional branch
- Absolute (register) unconditional jump
- A set of procedure calls that record a return link address in a general register.

Branch and Jump – Examples

- J Unconditional Jump

```
J name          ;PC ← name
```

- JAL Jump and Link

```
JAL name        ;R31 ← PC+4; PC ← name
```

- JALR Jump and Link Register

```
JALR R4         ;R31 ← PC+4; PC ← R4
```

Branch and Jump – Examples

- JR Jump Register

```
JR R3                ;PC ← R3
```

- BEQZ Branch Equal Zero

```
BEQZ R4,name         ;IF (R4 = 0) then PC ← name
```

- BNE Branch Not Equal

```
BNE R3,R4,name ;IF (R3 != R4) then PC ← name
```

Miscellaneous

- MOVZ Conditional Move if Zero

```
MOVZ R1,R2,R3 ;IF (R3 = 0) then R1 ← R2
```

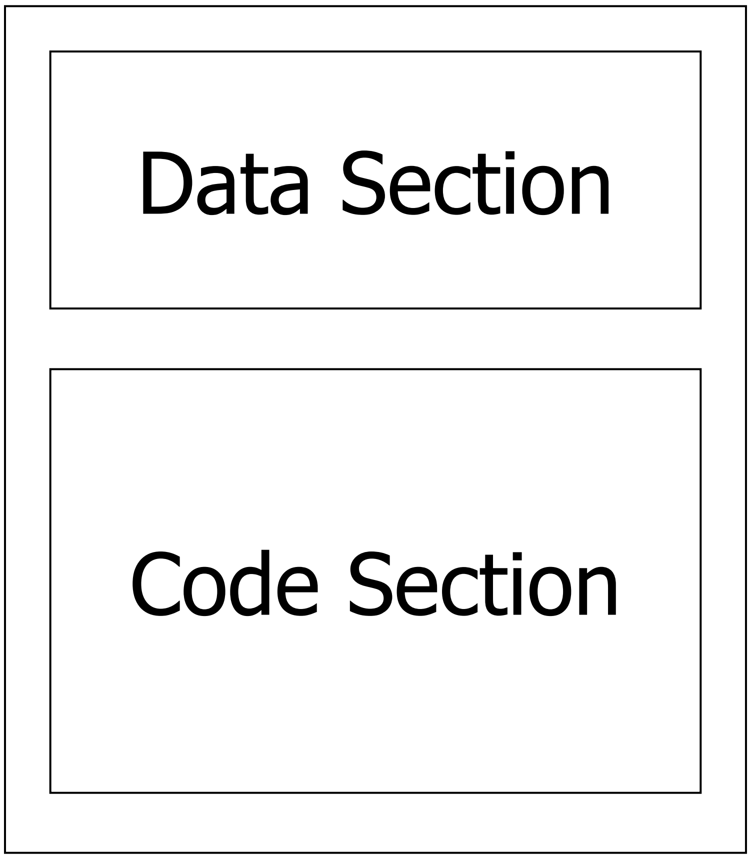
- NOP No Operation

```
NOP          ;It means  SLL R0, R0, 0
```

Floating Point

- The FPU instructions include almost the same instructions types:
  - Data Transfer Instructions
  - Arithmetic Instructions
  - Conditional Branch Instructions
  - Miscellaneous Instructions-

ASSEMBLER PROGRAMS



Assembler program

- Data Section
  - Variables
  - Constants
- Code Section
  - Program
  - Routines
  - Subroutines

Data Section

```
;******** MIPS64 INITIAL PROGRAM********
;---------------------------------------
; Program begin at symbol main
; requires module INPUT
;---------------------------------------

       .data
Prompt: .ascii  "An integer value >1:\0"

Vector: .word  1, 2, 3, 4, 5,

Result: .space  4
```

Program Title

Assembler Directives

Constants

Variables

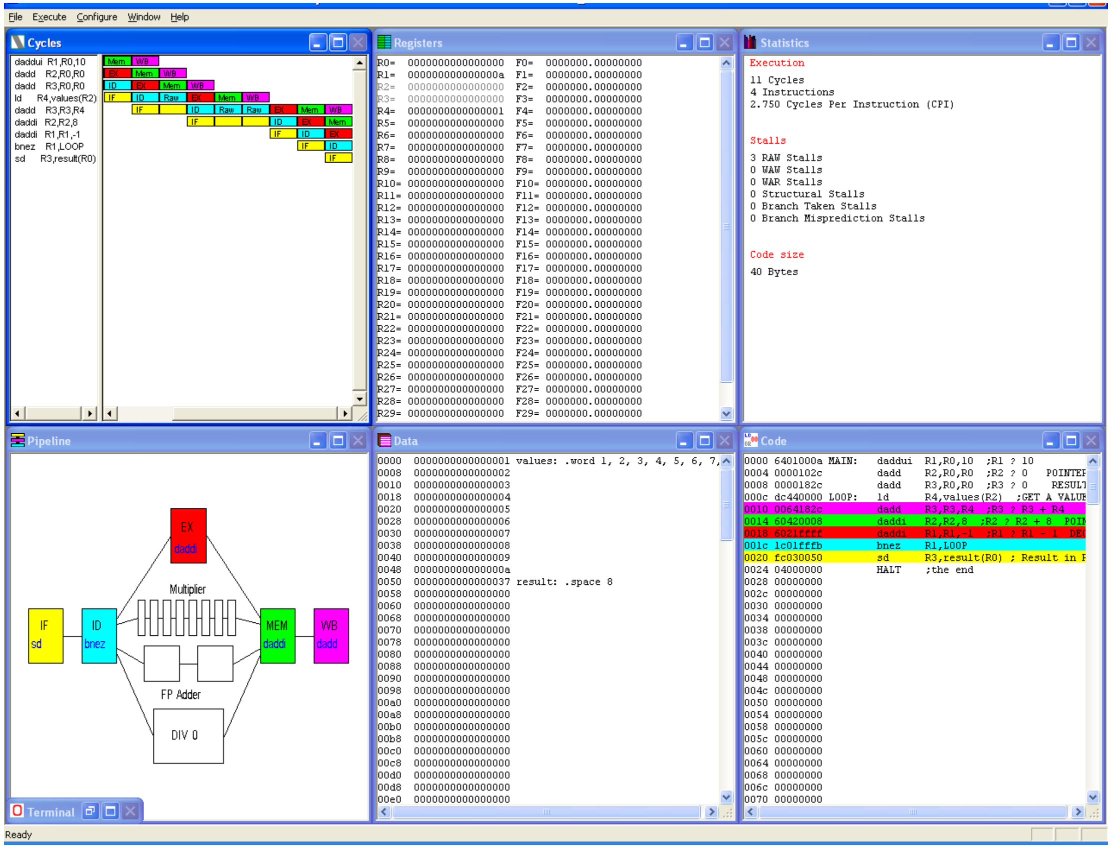
Code Section

```
        .Code
        .global   main

main:   addi      r1,r0,Info      ;*** Read value from stdin
        Jal       Input           into R1

                                  ;*** init values
        movi2fp   f10,r1          ;R1 -> D0    D0..Count
        cvti2d    f0,f10          register
        addi      r2,r0,1
        movi2fp   f11,r2          ;1 -> D2 D2..result
        cvti2d    f2,f11
        Movd      f4,f2           ;1-> D4  D4..Constant 1

Loop:   led       f0,f4           ;*** Break loop if D0 = 1
        bfpt      EndL            ;D0<=1 ?

        Multd     f2,f2,f0        ;*** Multiplication and
        subd      f0,f0,f4        next loop
        j         Loop

                                  ;*** write result to tdout

EndL:   sd        Print,f2
        addi      r14,r0,Print
        trap      5               ;*** end
```

- Assembler Directives
- Labels
- OPcode
- Operators
- Comments

WinMIPS64 the initial Glance

- Instruction set simulator
- 64-bit MIPS architecture
- Architectural features
  - Forwarding
  - Delay slot
  - Branch prediction.



WinMIPS64 the initial Glance

- Assembler Directives:
  - .data - start of data segment
  - .text - start of code segment
  - .code - start of code segment (same as .text)
  - .org <n> - start address
  - .space <n> - leave n empty bytes
  - .asciiz <s> - enters zero terminated ascii string
  - .ascii <s> - enter ascii string
  - .align <n> - align to n-byte boundary …

WinMIPS64 the initial Glance

- Assembler Directives:
  - .word <n1>,<n2>.. - enter word(s) of data (64-bits)
  - .byte <n1>,<n2>.. - enter bytes
  - .word32 <n1>,<n2>.. - enter 32 bit number(s)
  - .word16 <n1>,<n2>.. - enter 16 bit number(s)
  - .double <n1>,<n2>.. - enter floating-point number(s)

  where <n> denotes a number like 24, <s> denotes a string like "fred‟

  <n1>,<n2>.. denotes numbers seperated by commas.

Load and store

- lb - load byte
- lbu - load byte unsigned
- sb - store byte
- lh - load 16-bit half-word
- lhu - load 16-bit half word unsigned
- sh - store 16-bit half-word
- lw - load 32-bit word
- lwu - load 32-bit word unsigned
- sw - store 32-bit word
- ld - load 64-bit double-word
- sd - store 64-bit double-word
- l.d - load 64-bit floating-point
- s.d - store 64-bit floating-point

ALU operations

- daddi - add immediate
- daddui - add immediate unsigned
- andi - logical and immediate
- ori - logical or immediate
- xori - exclusive or immediate
- lui - load upper half of register immediate

Branches and Jumps

- j - jump to address
- jr - jump to address in register
- jal - jump and link to address (call subroutine)
- jalr - jump and link to address in register (call subroutine)
- beq - branch if pair of registers are equal
- bne - branch if pair of registers are not equal
- beqz - branch if register is equal to zero
- bnez - branch if register is not equal to zero

Floating Point

- add.d - add floating-point
- sub.d - subtract floating-point
- mul.d - multiply floating-point
- div.d - divide floating-point
- mov.d - move floating-point

Miscellaneous

- movz - move if register equals zero
- movn - move if register not equal to zero
- nop - no operation

A naïve example

- C = A + B

```
 .data
Val_A: .word 10
Val_B: .word 20
Val_C: .word 0

      .text
Main:
      ld R1, Val_A(R0)
      ld R2, Val_B(R0)
      dadd R3, R2, R1
      sd R3, Val_C(R0)
```

MIPS PROGRAM

```
 .data
Val_A: dw 10
Val_B: dw 20
Val_C: dw 0

…
Main:
       mov AX, Val_A
       add AX, Val_B
       mov Val_C, AX
```

8086 PROGRAM

A naïve example (1)

- C = A + B

```
  .data
Val_A: .word 10
Val_B: .word 20
Val_C: .word 0

       .text
Main:
       ld R1, Val_A(R0)
       ld R2, Val_B(R0)
       dadd R3, R2, R1
       sd R3, Val_C(R0)
```

Code Analysis	
# instructions	4
Code size [bytes]	16
Execution time [C.C.]	4

A naïve example (2)

- C = A + B

```
 .data
Val_A: dw 10
Val_B: dw 20
Val_C: dw 0

…
Main:
        mov AX, Val_A
        add AX, Val_B
        mov Val_C, AX
```

Code Analysis	
# Instructions	3
Code size [bytes]	8
Execution time [C.C.]	33

A 2nd example

```
     /* Sum of 10 integer values */
#include <stdio.h>

const long int values[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

int main() {
         long int result;

         result = 0;
         for (int i = 0; i < 10; i++) {

                  result = result + values[i];
         }


}
```

A 2nd example

```
;----------------------------
; Program: 10V_sum.s
; Sum of 10 integer values
;----------------------------
        .data
values: .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ;64-bit integers
result: .space 8

        .text
MAIN:   daddui  R1,R0,10  ;R1 ← 10
        dadd    R2,R0,R0  ;R2 ← 0   POINTER REG
        dadd    R3,R0,R0  ;R3 ← 0   RESULT REG

LOOP:   ld      R4,values(R2)      ;GET A VALUE IN R4
        dadd    R3,R3,R4  ;R3 ← R3 + R4
        daddi   R2,R2,8   ;R2 ← R2 + 8   POINTER INCREMENT
        daddi   R1,R1,-1  ;R1 ← R1 - 1   DECREMENT COUNTER
        bnez    R1,LOOP
        sd      R3,result(R0) ; Result in R3

        HALT              ;the end
```

References

- MIPS64™ Architecture For Programmers: Introduction to the MIPS64™ Architecture. Vol I, II, III. MIPS Technologies, Inc.
- WinMIPS64, Mike Scott

  http://indigo.ie/~mscott/