The DLX Architecture

The DLX is a hypothetical General Purpose Register Machine with Load/Store Architecture

The DLX is a **R**educed **I**nstruction **S**et **C**omputer:

- Its Architecture has been adapted to simple Instructions which, statistically, are mostly used in programs
- Complicated functions are simulated by several simple Instructions, i.e. by Software

The DLX considers measurements of these Instruction Sets:

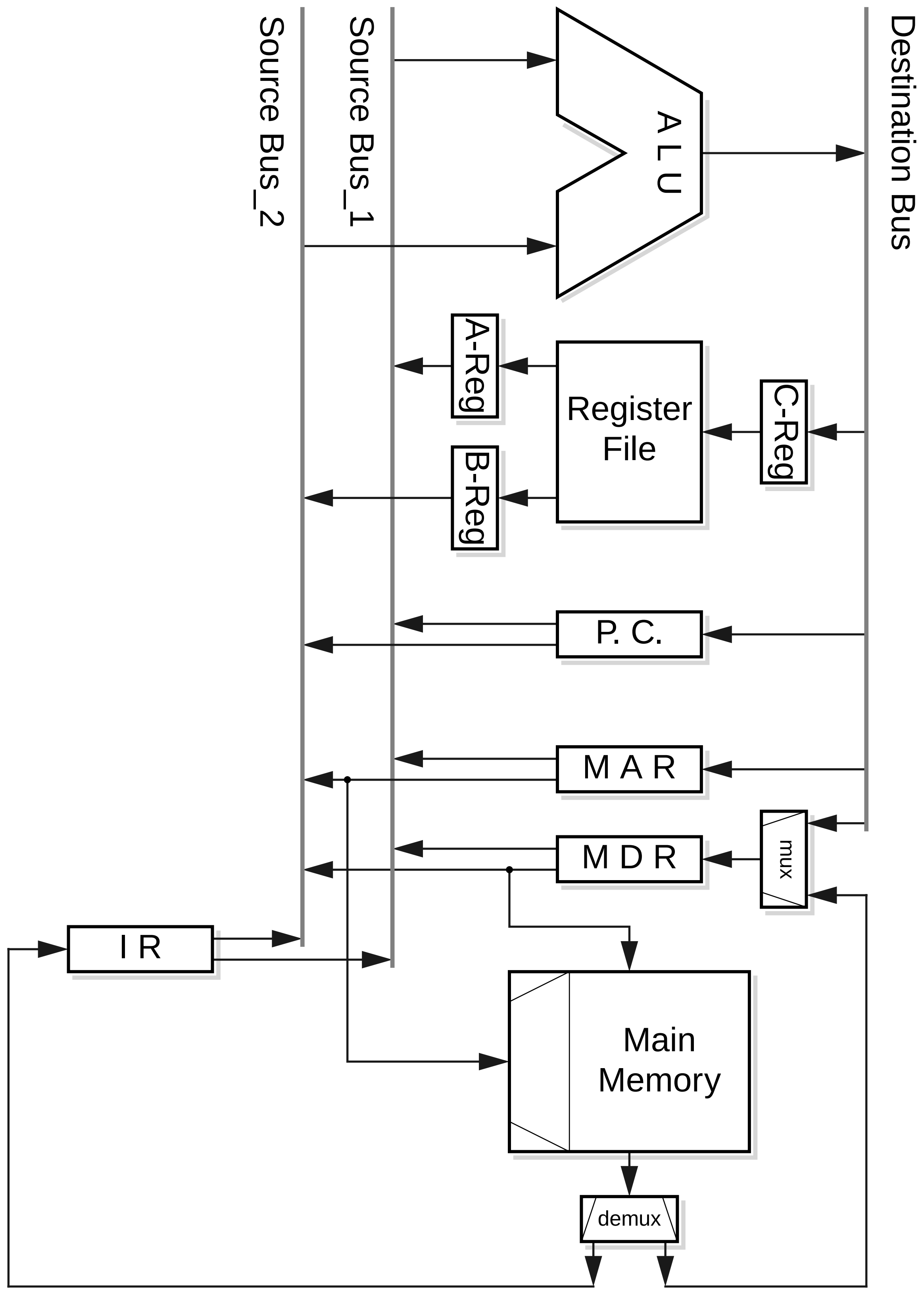
- DEC VAX Architecture
- IBM /370 Architecture
- Intel 8086 Architecture

Characteristics of the DLX-Machine:

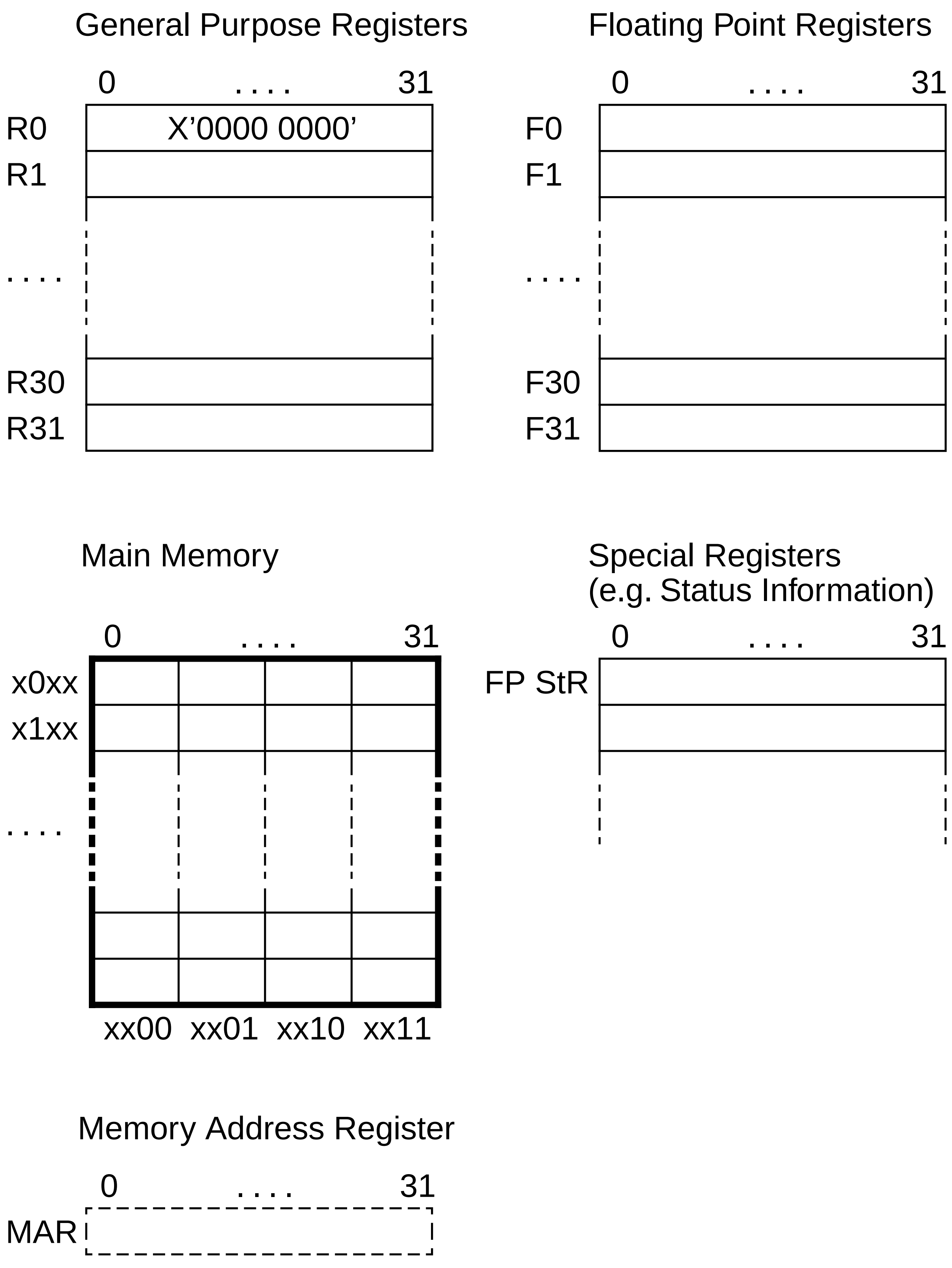
- Simple Load/Store Instruction Set
- General Purpose Register File
- Pipeline with High Efficiency
- Simple Decoding Scheme of the Instruction Set
- Compiler techniques with High Efficiency

<u>acc. to</u> D.A.Patterson, J.L. Hennessy: “Computer Architecture, a Quantitative Approach”, Morgan Kaufmann Publ., Inc. (1996)

DLX Processor Data Path

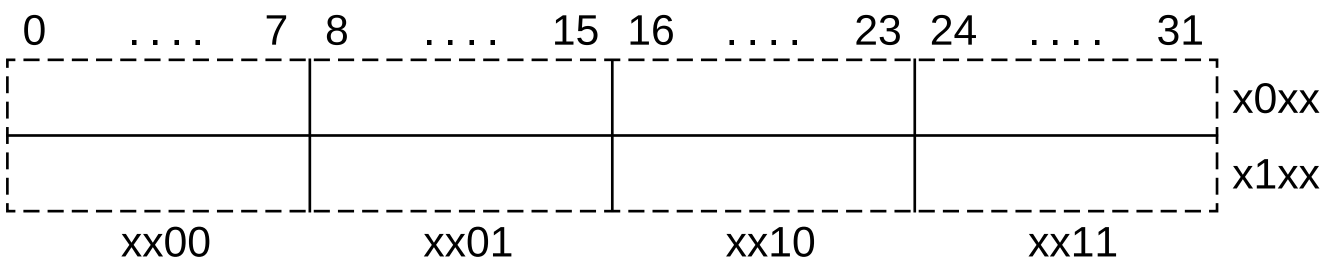


The Load/Store Machine DLX



DLX Data Formats

The Main Memory is Byte-addressed in “big endian” mode

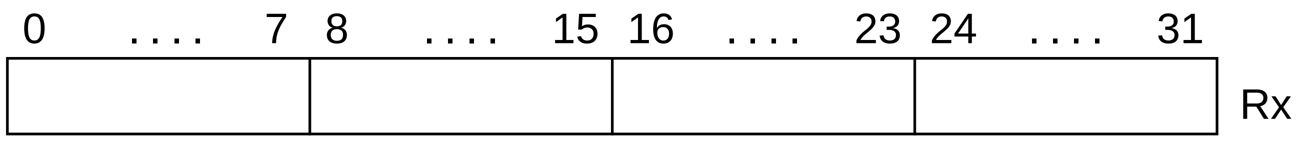


Data Formats are adjusted to the type boundary

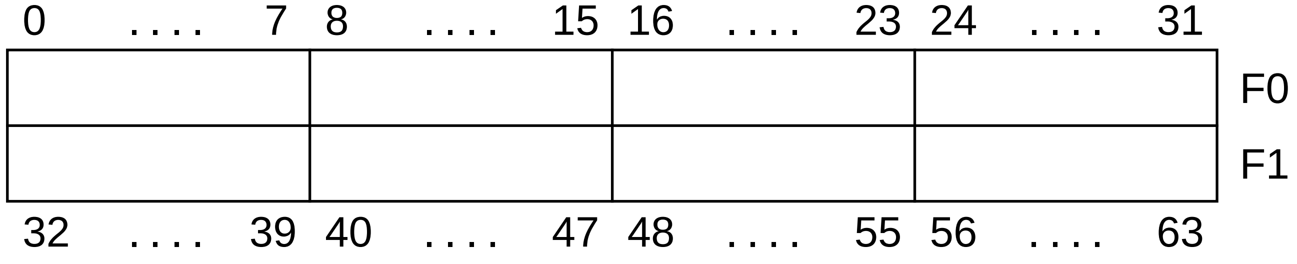
- 8 bit Byte
- 16 bit Halfword
- 32 bit Word
- 64 bit Doubleword

The Main Memory is accessed by Load/Store Instructions

- between Memory and General Purpose Registers acc. to Byte, Halfword or Word
  - Main Memory is adjusted to Byte or Halfword boundary
  - Load into the low-order part of the Register, the high-order part is filled acc. to the sign bit or with zeroes



- between Memory und Floating Point Registers with simple or double precision



DLX Instruction Formats

Instruction Format with three Operand Addresses

0 5	6 10	11....15	16 31
OPCode	OP1	OP2	OP3 or Immediate Data

- all Instructions are 32 bit long,
  adjusted to a Word boundary within Main Memory,
  Increasing of the Program Counter $PC \leftarrow PC + 4$
- 6 bit primary Operation Code (OPCode)
  $\rightarrow 2^6 = 64$ different Instructions

Instruction Classes

- Load/Store Operations
- Arithmetic/Logic Operations (ALU)
- Branches (conditional)
- Jumps (unconditional)
- Floating Point Operations

DLX Instruction Formats

Immediate Instruction Format

0 5	6 10	11....15	16	 31
OPCode	Rs1	Rd	S	Immediate Data/Offset

- Load or Store of Register Contents
- ALU-Operations with Immediate Data
- Branches (conditional):
  Branch on Zero, Branch on not Zero
- Jumps (unconditional):
  Jump Register, Jump and Link Register

Register-**R**egister Instruction Format

0 5	6 10	11....15	16....20	21 31
OPCode	Rs1	Rs2	Rd	OPC Extension

- ALU-Operations with Register Contents
- Transports between Special and General Purpose Registers

Jump Instruction Format

0 5	6	 31
OPCode	S	Offset added to PC + 4

- Jumps (unconditional): Jump, Jump and Link
- Call of the Operating System   Trap
- Return to User Program   RFE

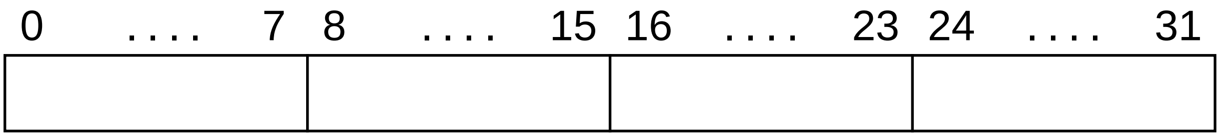
Load/Store Instructions

Immediate Instruction Format

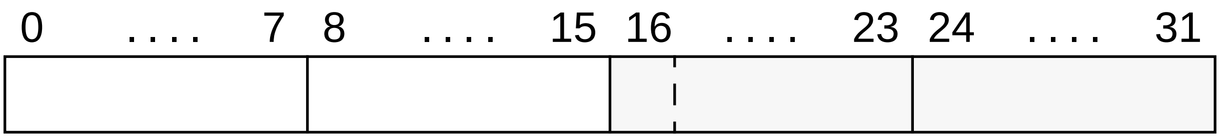
0 5	6 10	11....15	16	 31
OPCode	Rb	Rd	S	Offset (Displacement)

One Mode of Addressing:

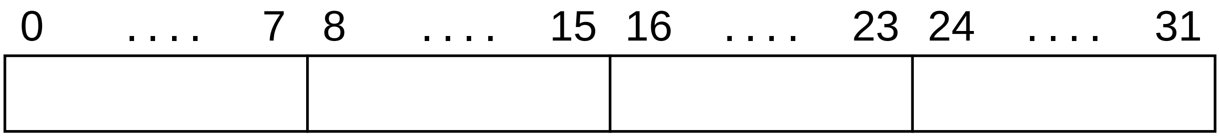
32 bit Base Register (Rb)



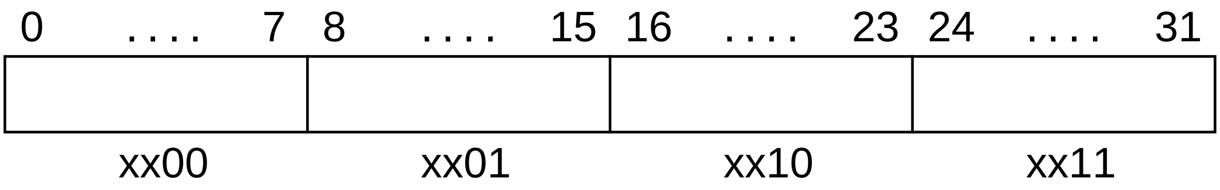
+ 16 bit Offset (Displacement) with Sign Bit Extension



= 32 bit MemoryAddress (MAR)

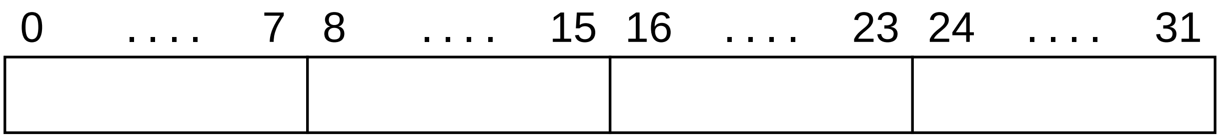


Byte-addressed Memory, 2^{32} = 4 GigaByte Address Space



Load or Store of Data Register (Rd)

- acc. to Byte, Halfword or Word



Stand: 8. Dezember 2002

Load/Store Instructions

- All Instructions use the same Addressing Mode
- They are available for all Data Types
- Values in Main Memory must be "aligned" to Type Boundary

Instr. Formats	Operations
`LW   R1,30(R2)`	$R1 \leftarrow_{32} M[30+R2]$
`LW   R1,90(R0)`	$R1 \leftarrow_{32} M[90+0]$
`LB   R1,40(R3)`	$R1 \leftarrow_{32} (M[40+R3]_0)^{24}\#\#M[40+R3]$
`LBU R1,40(R3)`	$R1 \leftarrow_{32} 0^{24}\#\#M[40+R3]$
`LH   R1,40(R3)`	$R1 \leftarrow_{32}$ $(M[40+R3]_0)^{16}\#\#M[40+R3]\#\#M[41+R3]$
`LHU R1,40(R3)`	$R1 \leftarrow_{32} 0^{16}\#\#M[40+R3]\#\#M[41+R3]$
`LF   F0,50(R3)`	$F0 \leftarrow_{32} M[50+R3]$
`LD   F0,50(R2)`	$F0\#\#F1 \leftarrow_{64} M[50+R2]$
`SW   50(R4),R3`	$M[50+R4] \leftarrow_{32} R3$
`SF   40(R3),F0`	$M[40+R3] \leftarrow_{32} F0$
`SD   40(R3),F0`	$M[40+R3] \leftarrow_{32} F0;$ $M[44+R3] \leftarrow_{32} F1$
`SH   52(R2),R3`	$M[52+R2] \leftarrow_{16} R3_{16..31}$
`SB   41(R3),R2`	$M[41+R3] \leftarrow_{8} R2_{24..31}$

ALU Instructions

Register-**R**egister Instruction Format

0 5	6 10	11....15	16....20	21 31
OPCode	Rs1	Rs2	Rd	OPC Extension

ALU-Operation defined by OPCode and Extension field:

- `ADD, SUB`
- `AND, OR, XOR`
- `SLL, SRL, SRA` (shift left, shift right)
- `Sxx` (compare & set register) `(LT,GT,LE,GE,EQ,NE)`

$$\boxed{Rd \leftarrow Rs1 \text{ ALU } Rs2}$$

32 bit Source Register (Rs1)

0 7	8 15	16 23	24 31

32 bit Source Register (Rs2)

0 7	8 15	16 23	24 31

32 bit Destination Register (Rd)

0 7	8 15	16 23	24 31

ALU Instructions

Immediate Instruction Format

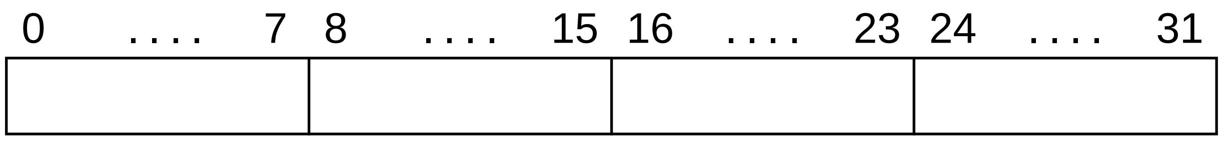
0 5	6 10	11....15	16	 31
OPCode	Rs1	Rd	S	Immediate Data

ALU-Operation defined by OPCode:

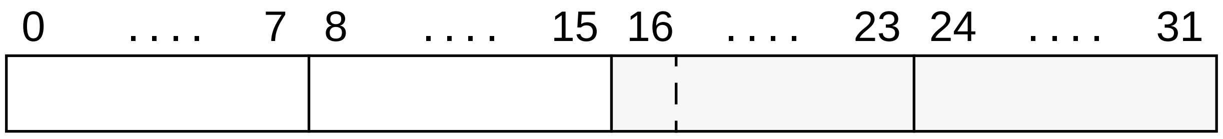
- `ADDI, SUBI`
- `SLLI, SRLI, SRAI` (shift left, shift right)
- `ANDI, ORI, XORI`
- `SxxI` (compare & set register) `(LT,GT,LE,GE,EQ,NE)`

$$\boxed{\text{Rd} \leftarrow \text{Rs1 ALU Immediate}}$$

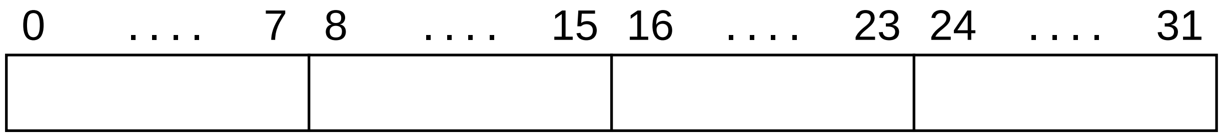
32 bit Source Register (Rs1)



16 bit Immediate Data with Sign Bit Extension



32 bit Destination Register (Rd)



ALU Instructions

- with Immediate Data (Instruction bits 16 . . 31) or
- only with Register Contents: RES, OP1, OP2 Addresses

Instructions	Instruction Formats	Operations
Add	`ADD R1,R2,R3`	R1 $\leftarrow$ R2+R3
Add immediate	`ADDI R1,R2,#3`	R1 $\leftarrow$ R2+3
Load high immediate	`LHI R1,#42`	R1 $\leftarrow$ 42##0^{16}
Shift left logical immediate	`SLLI R1,R2,#5`	R1 $\leftarrow$ R2<<5
Set less than	`SLT R1,R2,R3`	if (R2<R3) R1$\leftarrow$ 1 else R1$\leftarrow$ 0

Control Flow Instructions

The DLX Architecture contains

- two Branch Instructions (conditional): `BEQZ, BNEZ`
- Call of the Operating System: `TRAP`
- Return to User Program: `RFE`
- four Jump Instructions (unconditional):

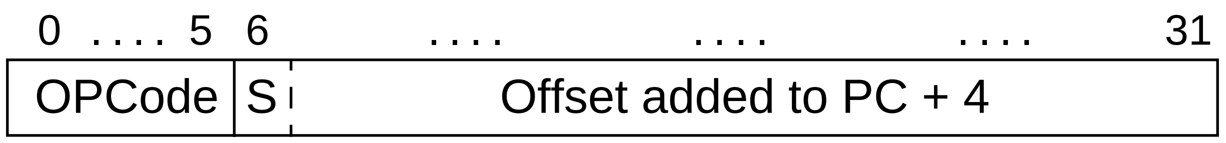
Destination Ad-	not Link	Link
PC relative	J	JAL
Register Content	JR	JALR

Link: Save content of Program Counter (Return Address) into R31 before jumping to a Sub-Program (Procedure Call)

Instruction Format	Operations
`J name`	PC $\leftarrow$ name; $(PC+4)-2^{25} \leq$ name $< (PC+4)+2^{25}$
`JR R3`	PC $\leftarrow$ R3
`JAL name`	R31 $\leftarrow$ PC+4; PC $\leftarrow$ name; $(PC+4)-2^{25} \leq$ name $< (PC+4)+2^{25}$
`JALR R2`	R31 $\leftarrow$ PC+4; PC $\leftarrow$ R2
`BEQZ R4,name`	if (R4==0) PC $\leftarrow$ name; $(PC+4)-2^{15} \leq$ name $< (PC+4)+2^{15}$
`BNEZ R4,name`	if (R4!=0) PC $\leftarrow$ name; $(PC+4)-2^{15} \leq$ name $< (PC+4)+2^{15}$

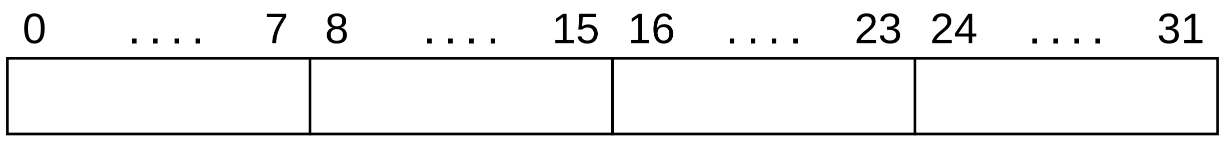
Jump Instructions

Jump Instruction Format

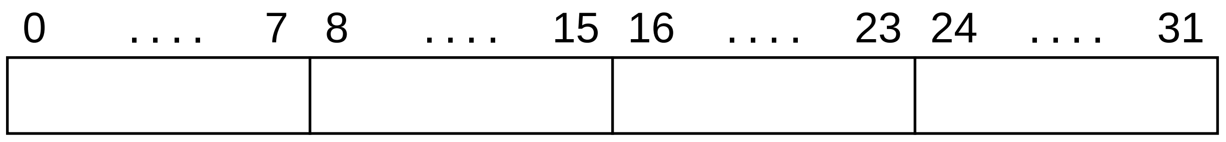


- Jump Instructions (unconditional):
  Jump (J), Jump and Link (JAL)
- Jump is relative to the Program Counter *(PC relative)*:

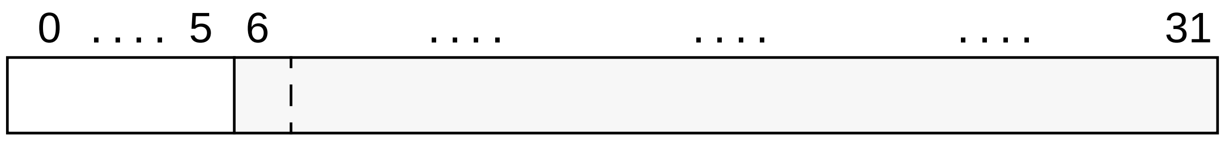
32 bit Program Counter



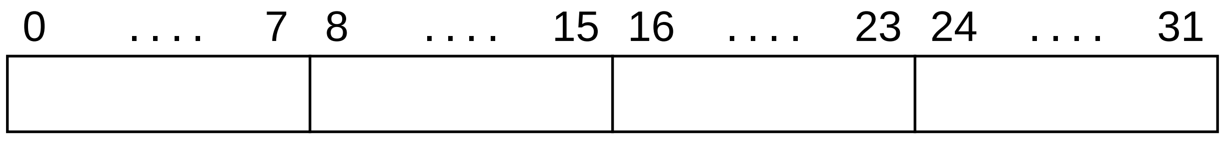
Increase the Program Counter $PC \leftarrow PC + 4;$
Link (only for JAL): $R31 \leftarrow PC$



+ 26 bit Offset (Displacement) with Sign Bit Extension



= Destination Addr. in Program Counter $PC \leftarrow PC + \text{Offset}$



Jump Instructions

Immediate Instruction Format

0 5	6 10	11....15	16 31
OPCode	Rs1	R0	X’0000’

- Jump Instructions (unconditional):
  Jump Register (JR), Jump and Link Register (JALR)
- Content of R0 = X’0000 0000’; Immediate = X’0000’
- The Destination Address is in the Source Register Rs1:

32 bit Program Counter (PC)

0 7	8 15	16 23	24 31

Increase the Program Counter $PC \leftarrow PC + 4;$
Link (only for JALR): $R31 \leftarrow PC$

0 7	8 15	16 23	24 31

32 bit General Purpose Register (Rs1)

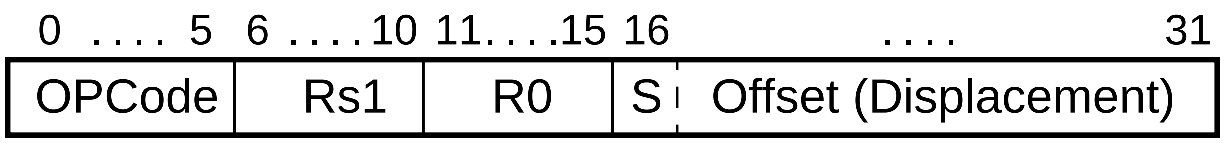
0 7	8 15	16 23	24 31

= Destination Addr. in Program Counter $PC \leftarrow$ Rs1

0 7	8 15	16 23	24 31

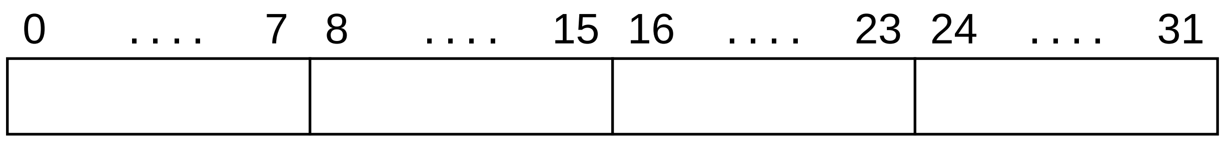
Branch Instructions

Immediate Instruction Format

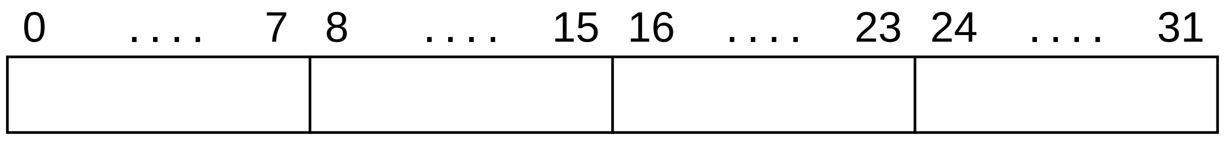


- Branch Instructions (conditional):
  Branch on Zero (`BEQZ`); Branch on Not Zero (`BNEZ`)
- The Source Register (Rs1) is tested:
  at `BEQZ`, if equal to zero; at `BNEZ`, if not equal to zero
- successful condition: Destination Address *PC* relative;
  unsuccessful condition: continue with next Instruction

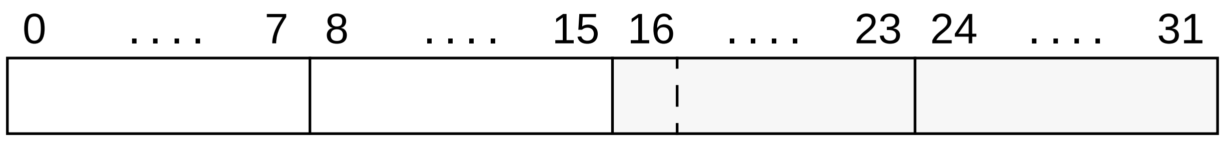
32 bit Program Counter (PC)



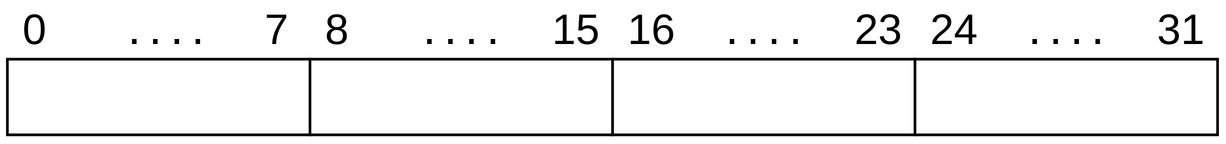
Increasing of the Program Counter $PC \leftarrow PC + 4$



+ 16 bit Offset (Displacement) with Sign Bit Extension



= Destination Addr. in Program Counter $PC \leftarrow PC + \text{Offset}$


0 7
8 15
16 23
24 31