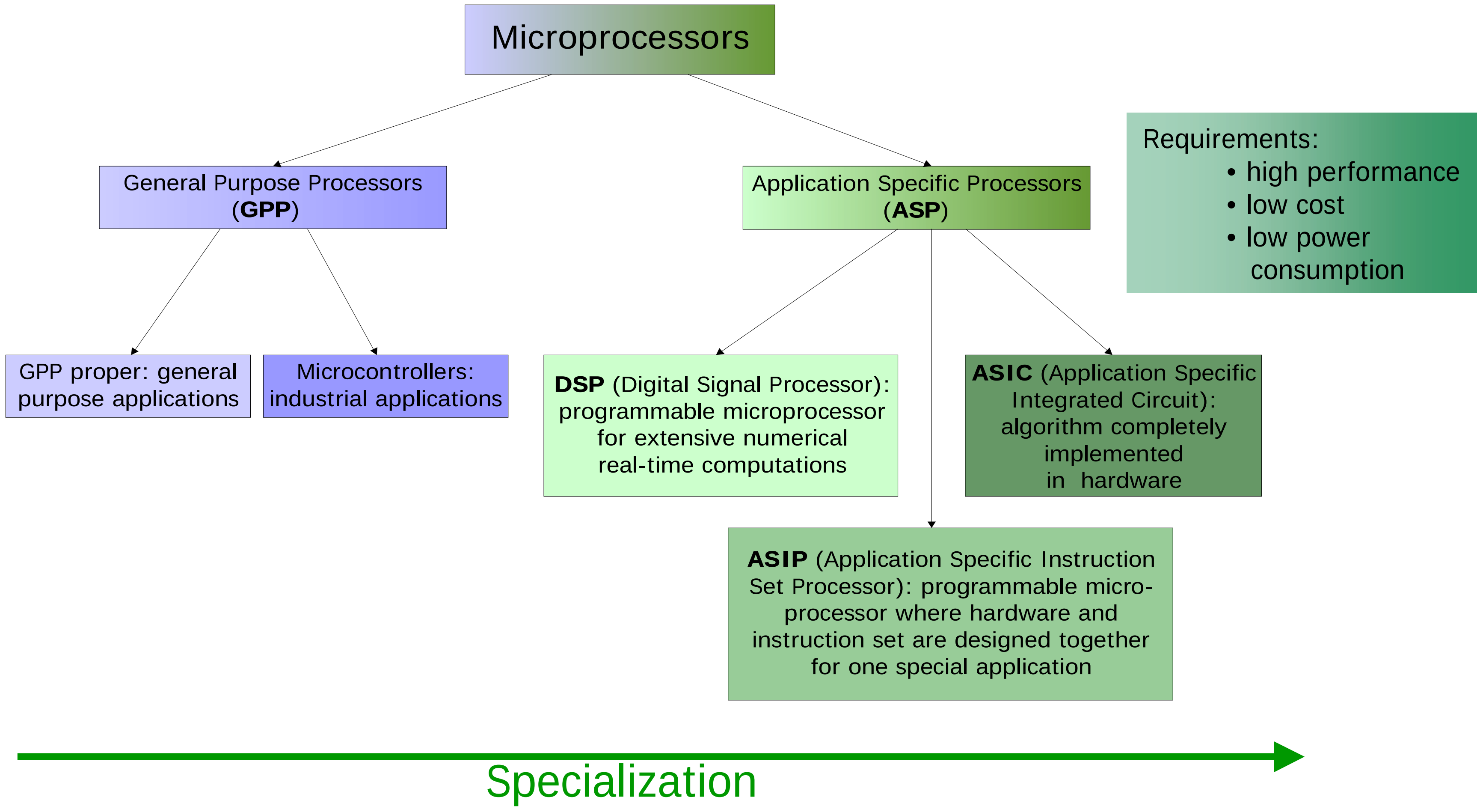
Classification of Microprocessors


Microprocessors
General Purpose Processors (**GPP**)
Application Specific Processors (**ASP**)
Requirements:
• high performance
• low cost
• low power consumption
GPP proper: general purpose applications
Microcontrollers: industrial applications
DSP (Digital Signal Processor): programmable microprocessor for extensive numerical real-time computations
ASIC (Application Specific Integrated Circuit): algorithm completely implemented in hardware
ASIP (Application Specific Instruction Set Processor): programmable micro-processor where hardware and instruction set are designed together for one special application
Specialization

DSP vs. GPP: Some Figures

Die Size	
DSP	3,9mm^2 – 60mm^2
GPP	≈100mm^2 – 345mm^2 (HP PA-RISC 8000)

Prices (1997)		
Fixed-Point DSPs	Motorola DSP56812	7$
	TI TMS320C54x	25$ - 40$
Floating-Point DSPs	TI TMSC44	130$
	ADSP 2106x SHARC	64$ - 358$
General Purpose Processors	Pentium 200 MHz	509$
	PowerPC 604e 225 MHz	620$

Focus: Digital Signal Processors

- Programmable microprocessors specialized for applications of digital signal processing.
- Characteristics:
  - Multiply-accumulate units
  - Multiple access memory architecture
  - Specialized addressing modes: auto-modify addressing, circular addressing, bit-reverse addressing
  - Residual control / predicated execution
  - Hardware loops / zero-overhead loops
  - Restricted interconnectivity between registers and functional units
  - Encoding restrictions

Overview

- Next central topic:
  - Hardware architecture fundamentals: Basics, example architecture: DLX
  - What is digital signal processing / why use DSPs?
  - Code generation: standard algorithms
  - Code generation: advanced algorithms

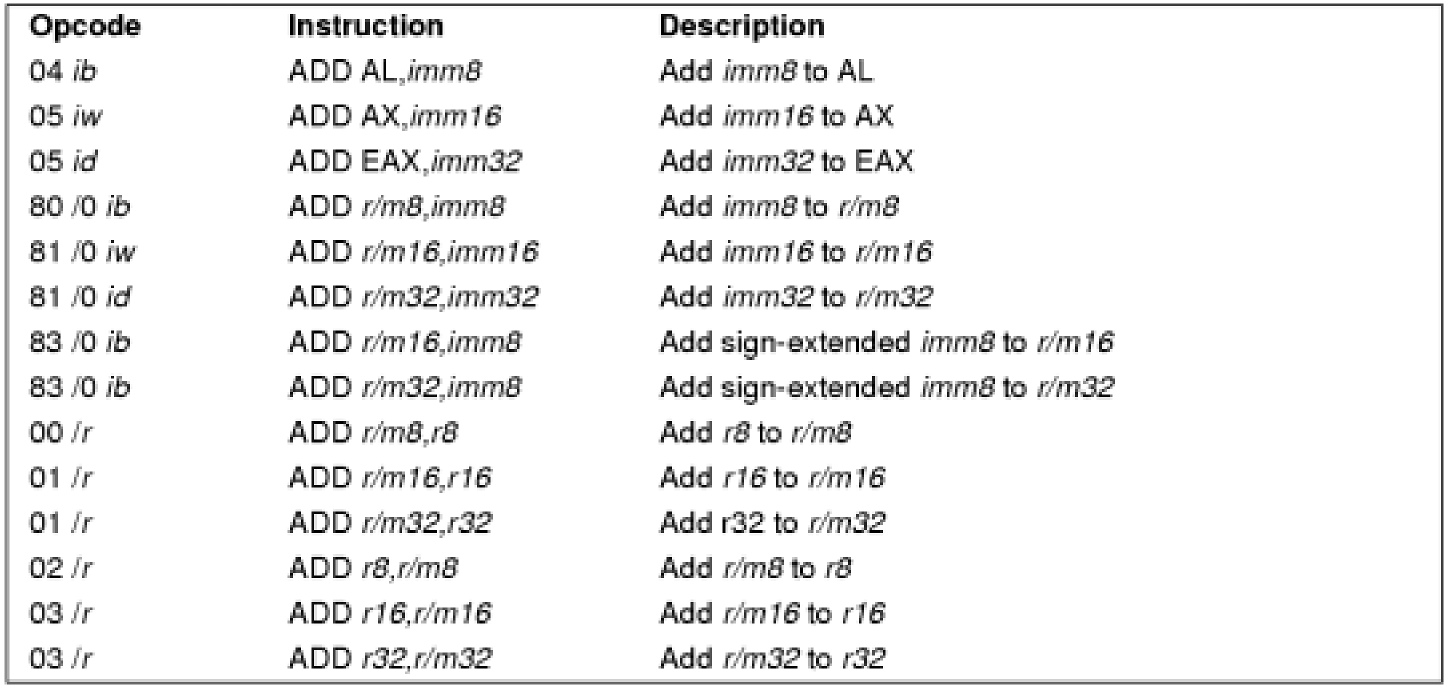
- After that (next year): How to ascertain that the generated code really does what it should?
  - Model checking
  - Static program analysis / abstract interpretation

Types of Microprocessors

- Complex Instruction Set Computer (CISC)
  - large number of complex addressing modes
  - many versions of instructions for different operands
  - different execution times for instructions
  - few processor registers
  - microprogrammed control logic

- Reduced Instruction Set Computer (RISC)
  - one instruction per clock cycle
  - memory accesses by dedicated load/store instructions
  - few addressing modes
  - hard-wired control logic

Example Instruction (IA-32)



Opcode	Instruction	Description
04 *ib*	ADD AL,*imm8*	Add *imm8* to AL
05 *iw*	ADD AX,*imm16*	Add *imm16* to AX
05 *id*	ADD EAX,*imm32*	Add *imm32* to EAX
80 /0 *ib*	ADD *r/m8,imm8*	Add *imm8* to *r/m8*
81 /0 *iw*	ADD *r/m16,imm16*	Add *imm16* to *r/m16*
81 /0 *id*	ADD *r/m32,imm32*	Add *imm32* to *r/m32*
83 /0 *ib*	ADD *r/m16,imm8*	Add sign-extended *imm8* to *r/m16*
83 /0 *ib*	ADD *r/m32,imm8*	Add sign-extended *imm8* to *r/m32*
00 /*r*	ADD *r/m8,r8*	Add *r8* to *r/m8*
01 /*r*	ADD *r/m16,r16*	Add *r16* to *r/m16*
01 /*r*	ADD *r/m32,r32*	Add r32 to *r/m32*
02 /*r*	ADD *r8,r/m8*	Add *r/m8* to *r8*
03 /*r*	ADD *r16,r/m16*	Add *r/m16* to *r16*
03 /*r*	ADD *r32,r/m32*	Add *r/m32* to *r32*

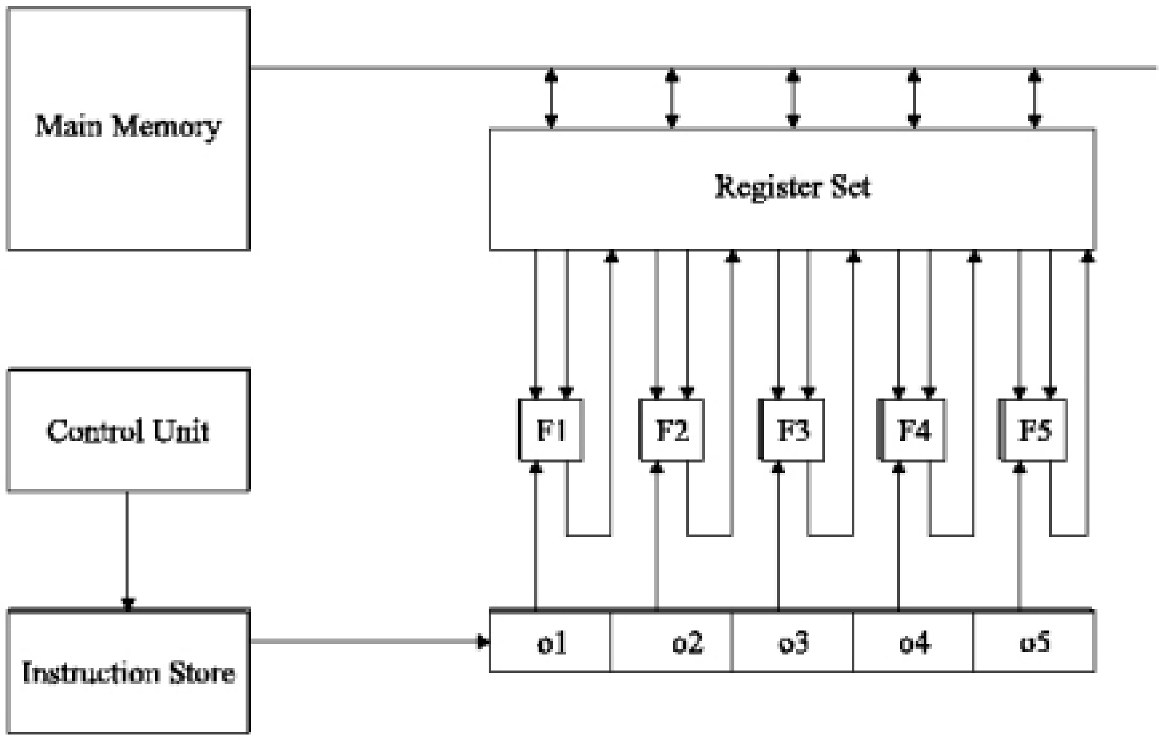
Execution time:
- 1 cycle: ADD EAX, EBX
- 2 cycles: ADD EAX, memvar32
- 3 cycles: ADD memvar32, EAX
- 4 cycles: ADD memvar16, AX

Instruction Width:
between 1 byte (NOP) and 16 bytes

Types of Microprocessors

- Very Long Instruction Word (VLIW)
  - statically determined instruction-level parallelism (under compiler control)
  - instructions are composed of different machine operations whose execution is started in parallel
  - many parallel functional units
  - large register sets
- Superscalar Processors
  - subclass of RISCs or CISCs
  - multiple instruction pipelines for overlapping execution of instructions
  - parallelism not necessarily exposed to the compiler

VLIW Architectures



VLIW Code Example

```
(* cycle 0 *)
IF r1   iaddi(0x2) r0 -> r38,   IF r1   isubi(0x4) r0 -> r7,
IF r1   isubi(0x3) r0 -> r8,    IF r1   isubi(0x2) r0 -> r36,
IF r1   isubi(0x1) r0 -> r37;

(* cycle 1 *)
IF r1   iaddi(0x3) r0 -> r39,   IF r1   uimm(0x190) -> r34,
IF r1   iaddi(0x8) r0 -> r35,   IF r1   h_st8d(0) r7 r6,
IF r1   h_st8d(4) r0 r6;
```

Types of Microprossesors

- Two classification criteria:
  - hardware characteristics
    - RISC
    - CISC
    - VLIW
    - Superscalar
  - characteristics of application areas
    - GPP (General Purpose Processor) / MCU (MicroController Unit)
    - SPP (Special Purpose Processor)
      - ASIC (Application-Specific Integrated Circuit)
      - ASIP (Application-Specific Instruction-set Processor)
      - DSP (Digital Signal Processor)

Hardware Design

- Instruction set architecture: interface of the processor to the user / compiler writer.
- Design goals:
  - Maximize the performance subject to a given cost limit, or
  - Minimize the cost subject to specified performance requirements for the application area.
- Performance depends on the application area; for evaluating the performance, representative benchmarks for that application area should be used.
- The achievable performance of a processor depends on the quality of the compiler / code generator.

Classical RISC: DLX

- In this lecture: Simplified DLX with instruction set encoding adapted from the MIPS R2000: [*Mueller,Paul. Computer Architecture. Complexity and Correctness. 2000*].
- DLX: RISC architecture with 3 instruction formats.
- 32 general purpose registers GPR[31:0]; GPR[0] is always 0.
- Memory accesses only by load/store instructions that move data between the general purpose registers and the memory M.
- Single addressing mode: effective address *ea* is the sum of a register and an immediate constant.
- Except for shifts, immediate constants are always sign extended.

DLX: Instruction Formats

- I-Type: Standard layout for instructions with an immediate operand

6	5	5	16
opcode	RS1	RD	immediate

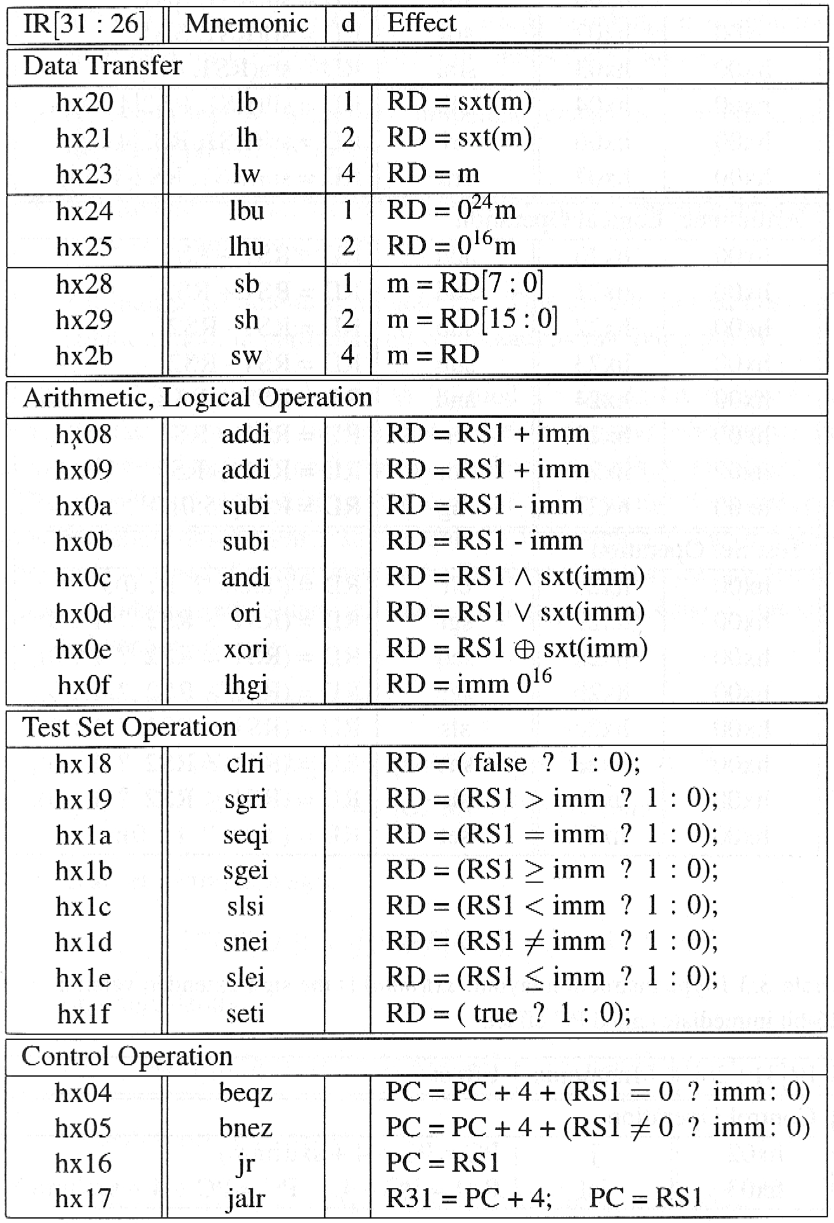
- J-type: control instructions.

6	26
opcode	PC offset

- R-type:

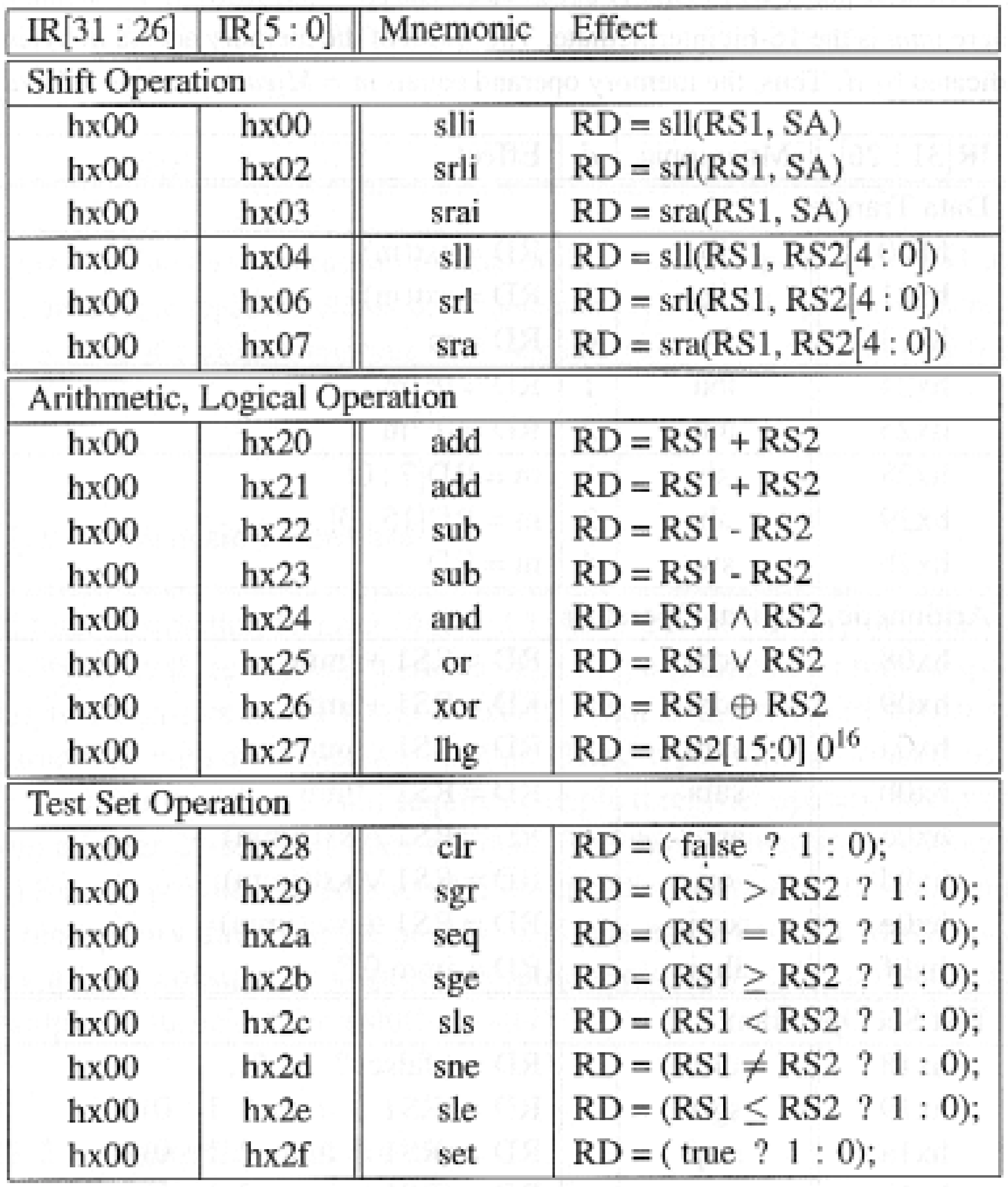
6	5	5	5	5	6
opcode	RS1	RS2	RD	SA	function

DLX: Instruction Set Encoding: I-Type Instructions



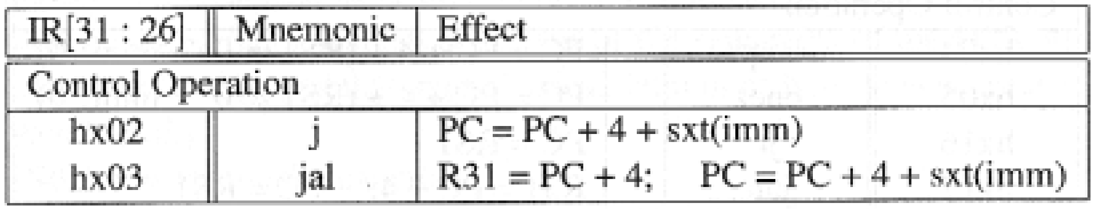
IR[31 : 26]	Mnemonic	d	Effect
Data Transfer			
hx20	lb	1	RD = sxt(m)
hx21	lh	2	RD = sxt(m)
hx23	lw	4	RD = m
hx24	lbu	1	RD = 0^{24}m
hx25	lhu	2	RD = 0^{16}m
hx28	sb	1	m = RD[7 : 0]
hx29	sh	2	m = RD[15 : 0]
hx2b	sw	4	m = RD
Arithmetic, Logical Operation			
hx08	addi		RD = RS1 + imm
hx09	addi		RD = RS1 + imm
hx0a	subi		RD = RS1 - imm
hx0b	subi		RD = RS1 - imm
hx0c	andi		RD = RS1 $\wedge$ sxt(imm)
hx0d	ori		RD = RS1 $\vee$ sxt(imm)
hx0e	xori		RD = RS1 $\oplus$ sxt(imm)
hx0f	lhgi		RD = imm 0^{16}
Test Set Operation			
hx18	clri		RD = (false ? 1 : 0);
hx19	sgri		RD = (RS1 $>$ imm ? 1 : 0);
hx1a	seqi		RD = (RS1 $=$ imm ? 1 : 0);
hx1b	sgei		RD = (RS1 $\geq$ imm ? 1 : 0);
hx1c	slsi		RD = (RS1 $<$ imm ? 1 : 0);
hx1d	snei		RD = (RS1 $\neq$ imm ? 1 : 0);
hx1e	slei		RD = (RS1 $\leq$ imm ? 1 : 0);
hx1f	seti		RD = (true ? 1 : 0);
Control Operation			
hx04	beqz		PC = PC + 4 + (RS1 $=$ 0 ? imm : 0)
hx05	bnez		PC = PC + 4 + (RS1 $\neq$ 0 ? imm : 0)
hx16	jr		PC = RS1
hx17	jalr		R31 = PC + 4; PC = RS1

DLX: Instruction Set Encoding: R-Type Instructions



IR[31 : 26]	IR[5 : 0]	Mnemonic	Effect
Shift Operation			
hx00	hx00	slli	RD = sll(RS1, SA)
hx00	hx02	srli	RD = srl(RS1, SA)
hx00	hx03	srai	RD = sra(RS1, SA)
hx00	hx04	sll	RD = sll(RS1, RS2[4 : 0])
hx00	hx06	srl	RD = srl(RS1, RS2[4 : 0])
hx00	hx07	sra	RD = sra(RS1, RS2[4 : 0])
Arithmetic, Logical Operation			
hx00	hx20	add	RD = RS1 + RS2
hx00	hx21	add	RD = RS1 + RS2
hx00	hx22	sub	RD = RS1 - RS2
hx00	hx23	sub	RD = RS1 - RS2
hx00	hx24	and	RD = RS1 $\wedge$ RS2
hx00	hx25	or	RD = RS1 $\vee$ RS2
hx00	hx26	xor	RD = RS1 $\oplus$ RS2
hx00	hx27	lhg	RD = RS2[15:0] 0^{16}
Test Set Operation			
hx00	hx28	clr	RD = (false ? 1 : 0);
hx00	hx29	sgr	RD = (RS1 $>$ RS2 ? 1 : 0);
hx00	hx2a	seq	RD = (RS1 $=$ RS2 ? 1 : 0);
hx00	hx2b	sge	RD = (RS1 $\geq$ RS2 ? 1 : 0);
hx00	hx2c	sls	RD = (RS1 $<$ RS2 ? 1 : 0);
hx00	hx2d	sne	RD = (RS1 $\neq$ RS2 ? 1 : 0);
hx00	hx2e	sle	RD = (RS1 $\leq$ RS2 ? 1 : 0);
hx00	hx2f	set	RD = (true ? 1 : 0);

DLX: Instruction Set Encoding: J-Type Instructions

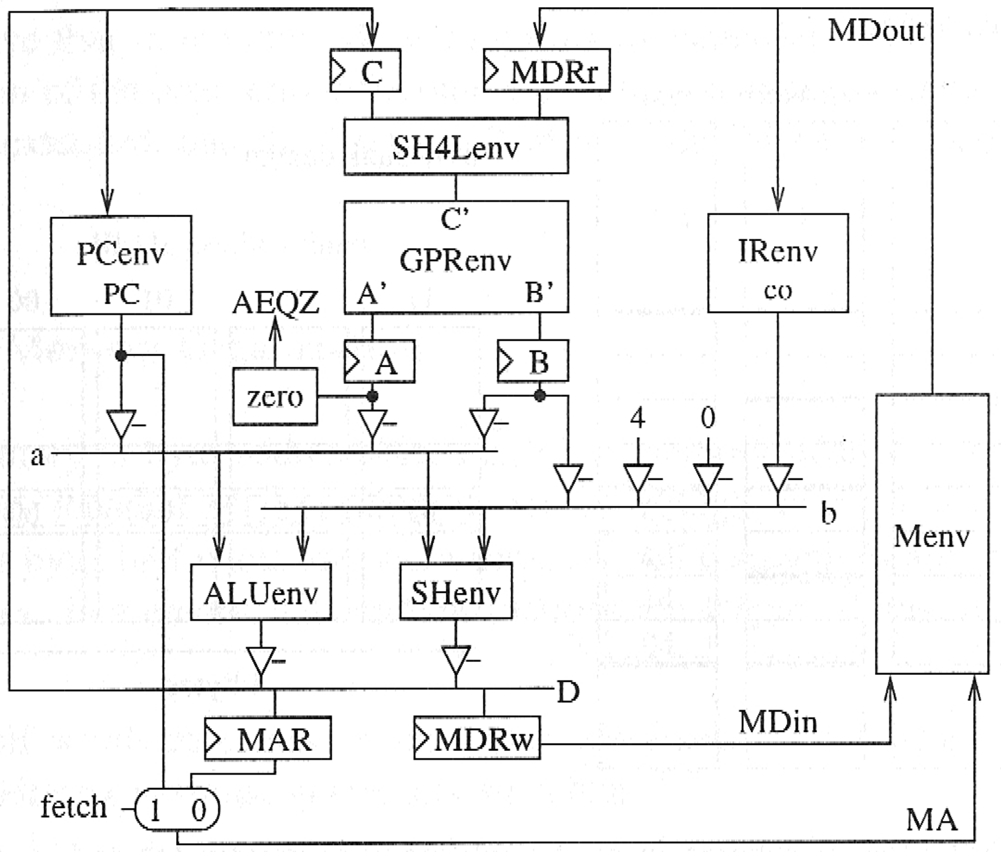


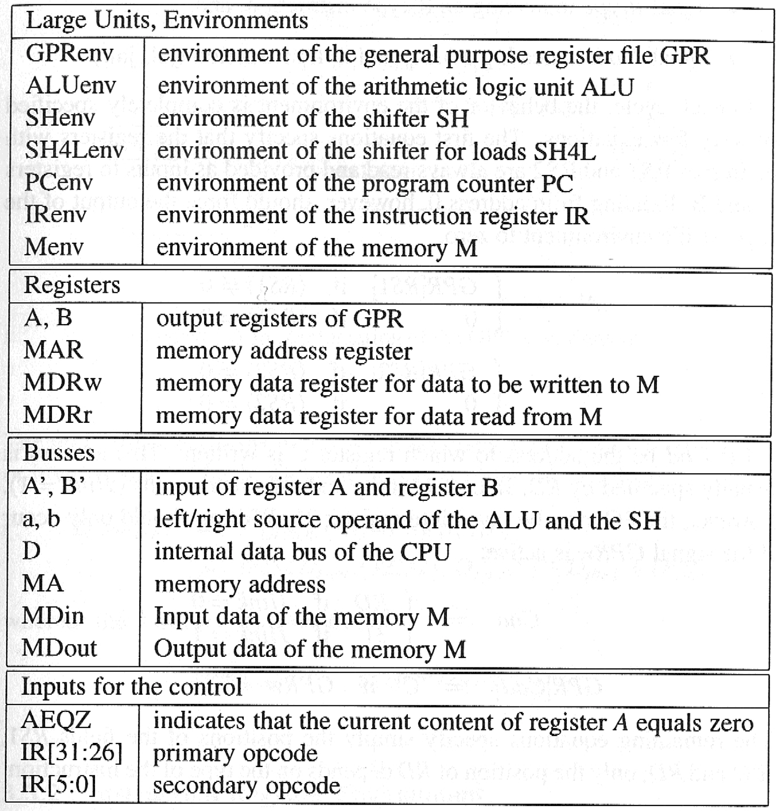
IR[31 : 26]	Mnemonic	Effect
Control Operation		
hx02	j	PC = PC + 4 + sxt(imm)
hx03	jal	R31 = PC + 4; PC = PC + 4 + sxt(imm)

DLX: Memory Organization

- Memory is byte addressable and performs byte, half word, and word accesses.
- Alignment restrictions:
  - half words h must have even (byte) addresses: $h[15:0] = M[e+1:e]$
  - words w must have (byte) addresses divisible by four: $w[31:0] = M[e+3:e]$
- Memory word with address e: $Mword[e] = M[e+3:e]$.
- The bytes of words are numbered in little endian order, i.e.
  - $byte_j(w) = w[8j+7:8j]$.
  - $byte_{[i:j]}(w) = byte_i(w) \ldots byte_j(w)$

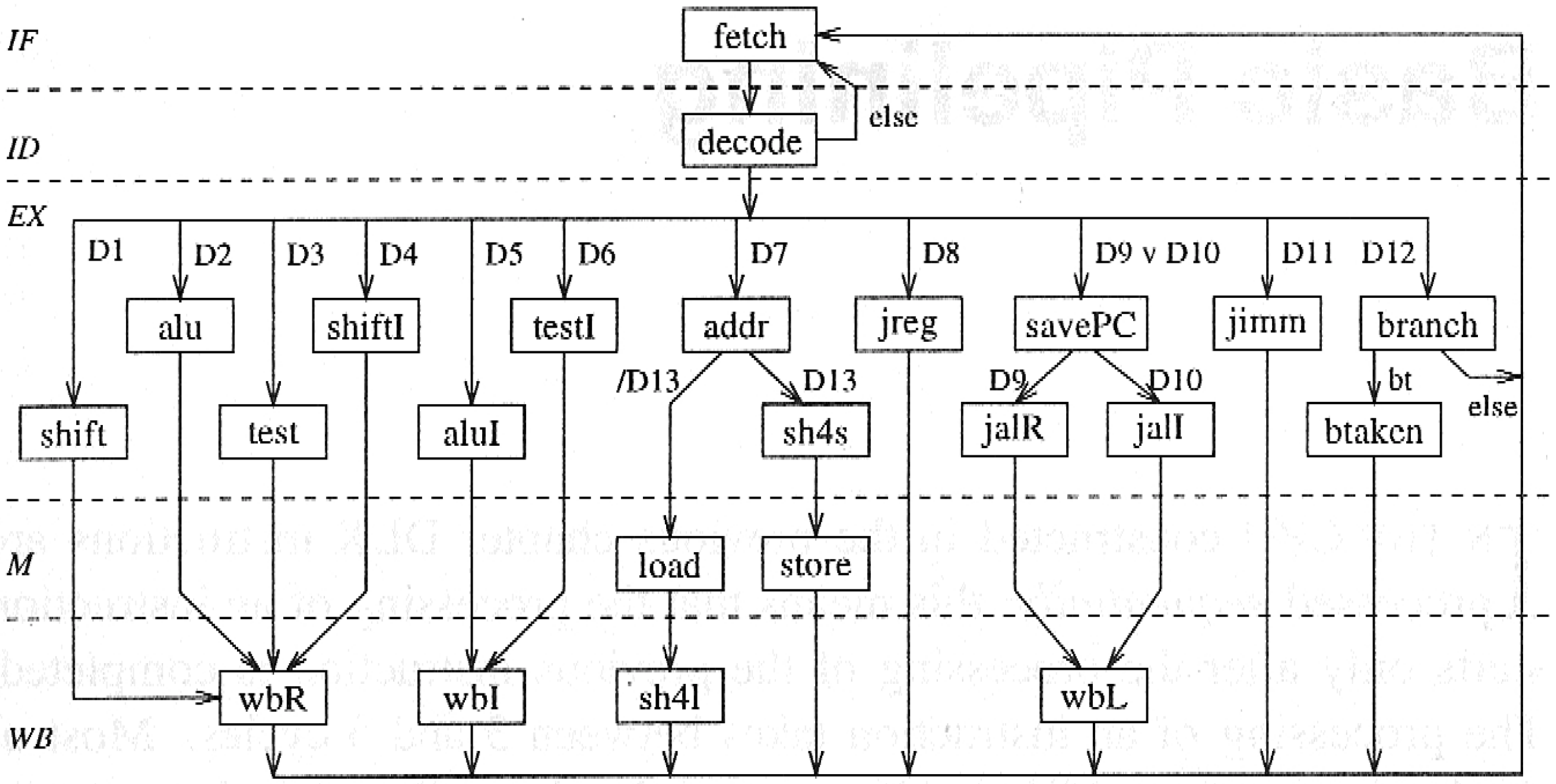
DLX: Data Paths



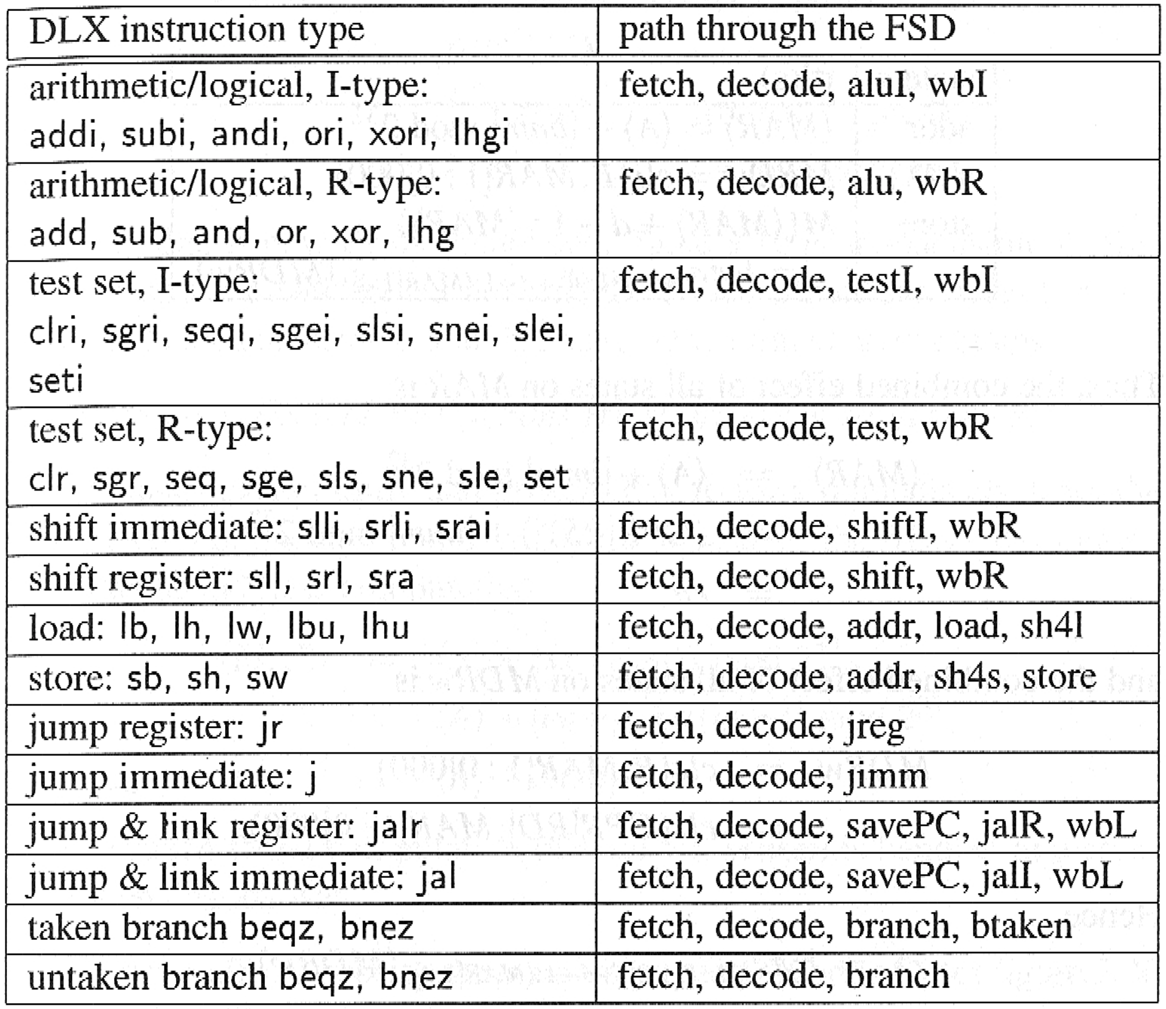


Large Units, Environments	
GPRenv	environment of the general purpose register file GPR
ALUenv	environment of the arithmetic logic unit ALU
SHenv	environment of the shifter SH
SH4Lenv	environment of the shifter for loads SH4L
PCenv	environment of the program counter PC
IRenv	environment of the instruction register IR
Menv	environment of the memory M
Registers	
A, B	output registers of GPR
MAR	memory address register
MDRw	memory data register for data to be written to M
MDRr	memory data register for data read from M
Busses	
A', B'	input of register A and register B
a, b	left/right source operand of the ALU and the SH
D	internal data bus of the CPU
MA	memory address
MDin	Input data of the memory M
MDout	Output data of the memory M
Inputs for the control	
AEQZ	indicates that the current content of register *A* equals zero
IR[31:26]	primary opcode
IR[5:0]	secondary opcode

DLX: FSD and Pipeline

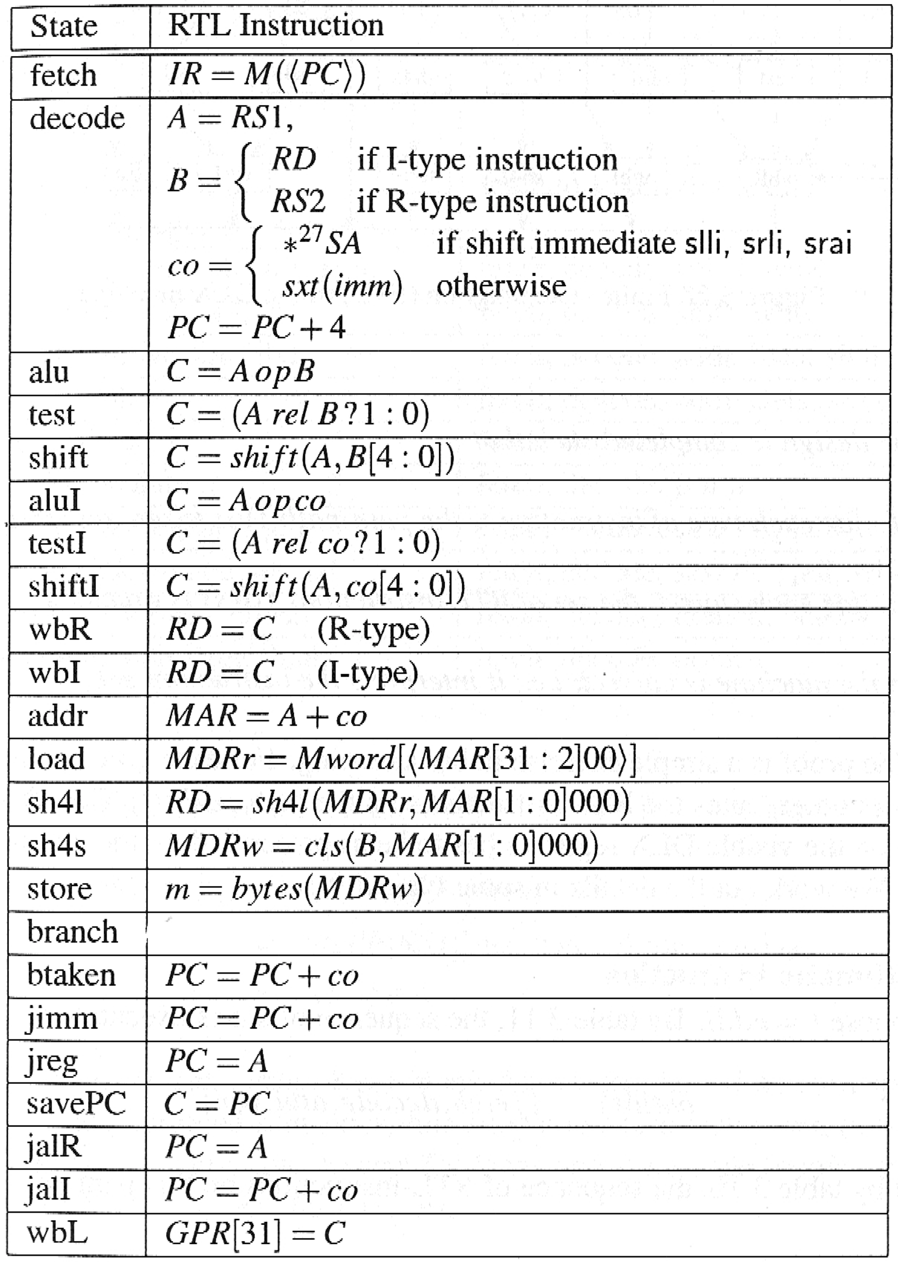


DLX: FSD Paths of Instructions



DLX instruction type	path through the FSD
arithmetic/logical, I-type: addi, subi, andi, ori, xori, lhgi	fetch, decode, aluI, wbI
arithmetic/logical, R-type: add, sub, and, or, xor, lhg	fetch, decode, alu, wbR
test set, I-type: clri, sgri, seqi, sgei, slsi, snei, slei, seti	fetch, decode, testI, wbI
test set, R-type: clr, sgr, seq, sge, sls, sne, sle, set	fetch, decode, test, wbR
shift immediate: slli, srli, srai	fetch, decode, shiftI, wbR
shift register: sll, srl, sra	fetch, decode, shift, wbR
load: lb, lh, lw, lbu, lhu	fetch, decode, addr, load, sh4l
store: sb, sh, sw	fetch, decode, addr, sh4s, store
jump register: jr	fetch, decode, jreg
jump immediate: j	fetch, decode, jimm
jump & link register: jalr	fetch, decode, savePC, jalR, wbL
jump & link immediate: jal	fetch, decode, savePC, jalI, wbL
taken branch beqz, bnez	fetch, decode, branch, btaken
untaken branch beqz, bnez	fetch, decode, branch

DLX: Instruction Semantics in Register Transfer Language (RTL)



State	RTL Instruction
fetch	$IR = M(\langle PC\rangle)$
decode	$A = RS1$, $B = \begin{cases} RD & \text{if I-type instruction} \\ RS2 & \text{if R-type instruction} \end{cases}$ $co = \begin{cases} *^{27}SA & \text{if shift immediate slli, srli, srai} \\ sxt(imm) & \text{otherwise} \end{cases}$ $PC = PC + 4$
alu	$C = A\,op\,B$
test	$C = (A\ rel\ B\,?\,1:0)$
shift	$C = shift(A, B[4:0])$
aluI	$C = A\,op\,co$
testI	$C = (A\ rel\ co\,?\,1:0)$
shiftI	$C = shift(A, co[4:0])$
wbR	$RD = C$ (R-type)
wbI	$RD = C$ (I-type)
addr	$MAR = A + co$
load	$MDRr = Mword[\langle MAR[31:2]00\rangle]$
sh4l	$RD = sh4l(MDRr, MAR[1:0]000)$
sh4s	$MDRw = cls(B, MAR[1:0]000)$
store	$m = bytes(MDRw)$
branch	
btaken	$PC = PC + co$
jimm	$PC = PC + co$
jreg	$PC = A$
savePC	$C = PC$
jalR	$PC = A$
jalI	$PC = PC + co$
wbL	$GPR[31] = C$

DLX: Pipelining

cycles

IF	I0	I1	I2	I3	I4	…	…	…	…
ID		I0	I1	I2	I3	I4	…	…	…
EX			I0	I1	I2	I3	I4	…	…
M		idle		I0	I1	I2	I3	I4	…
WB					I0	I1	I2	I3	I4