05-3

The Useful Features

- Lots of general purpose registers.
- Integer and floating-point operands.
- Basic arithmetic and logical operations.
- Basic addressing modes: register, immediate, displacement.
- Adequate immediate and displacement sizes.
- Etc.

Simple, High-Speed Implementation

- Load-Store Architecture: ALU instructions do not access memory.
- Simple Coding: uniform instruction sizes, few instruction types.
- Work Balance: Instructions do about the same amount of work.
- Separate integer and FP register files.

EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-3

05-1

DLX ISA

Coverage

- Textbook Section 2.8

Topics

- DLX Goals
- DLX Instruction Highlights
- DLX Instruction Coding
- Synthetic Instructions (NIB)

EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-1

05-4

Simple Coding Advantages

- Simpler and faster decoding logic.
- Execution can start before decoding complete.

Work Balance Advantages

- Efficient use of CPU hardware.
  - Integer operations are balanced.
  - Floating-point operations are not. (Division takes longer than ADD.)

EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-4

05-2

DLX Goals

- A typical RISC processor.
- Incorporate features with demonstrated usefulness.
- Enable simple, high-speed, implementation

Demonstration of Usefulness of Features

- Covered in Chapters 1, 2.
- Determined by analyzing existing ISAs.
- Some usefulness illustrated with graphs (e.g., immediate sizes).
- Less useful, “it would be nice,” features omitted.

EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-2

LHI Examples

Used to load constants.

Needed because immediate size limited to 16 bits.

Example, set r1 = 0x12345678

```
LHI r1, 0x1234          ! r1 = 0x12340000
ORI r1, r1, #0x5678     ! r1 = r1 | 0x5678
```


Separate Register File Advantages

Double the number of registers with only 1 bit per instruction (in opcode).

(Otherwise, 1 extra bit per operand would be needed.)

Splits register reads and writes between two register files.

With one large set of registers ...
... if n instructions start at once, need to access 2n registers ...
... all stored in one file (memory device) — expensive and slow.

With separate integer and FP register files ...
... each file would only have to provide n registers ...
... (assuming equal number of integer and FP instructions).

Note: Currently, n varies from 2 to 4.

Details on these implementation factors covered later.

Fun With r0, and other tricks.

Set a register to zero:

```
ADD  r1, r0, r0    ! r1 = 0
ADDI r1, r0, #0
SUB  r1, r1, r1
XOR  r1, r1, r1
```

Move one register to another:

```
ADD  r2, r1, r0    ! r2 = r1
AND  r2, r1, r1
ADDI r2, r1, #0
```

Bitwise Negation

```
XORI r2, r1, #-1  ! r2 =  r1
```


DLX Instruction Highlights

For detailed instruction descriptions, see text.

Instruction Highlights

Single, but flexible, memory addressing mode: Displacement.

Special load high (LHI) instruction for (part of) 32-bit constants.

Dummy, but very handy, register r0. (Value always 0.)

Displacement Addressing Flexibility

Classic Displacement Addressing

```
LW r1, 4(r2)     ! r1 = MEM[ r2 + 4 ]
```

Register Indirect (Use zero displacement.)

```
LW r1, 0(r2)     ! r1 = MEM[ r2 ]
```

Absolute (Use r0, limited because of immediate size.)

```
LW r1, 1234(r0)  ! r1 = MEM[ 1234 ]
```

05-11

Type I:

Fields: Opcode 6, rs1 5, rd 5, immediate 16.

Used for loads, stores, some CTIs, and ALU immediate instructions.

Examples

```
ADDI r1, r2, #3    ! r1 = r2 + 3
LW   r2, 10(r3)    ! r2 = MEM[r3+10]
BEQZ r1, 20        ! if( r1 == 0 ) goto PC + 4 + 20  (rd unused).
JR   r1            ! goto r1
JALR r1            ! r31 = pc + 4; goto r1
```


EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-11

05-9

DLX Instruction Coding

All instructions have 6-bit opcode.

Three types.

Type R: Three registers, plus extra opcode field.

Type I: Two registers, plus 16-bit immediate field.

Type J: One 26-bit immediate field.

EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-9

05-12

Type J:

Fields: Opcode 6, offset 26

Used for jump and jump & link.

Examples:

J 0x1234 ! goto PC + 4 + 0x1234

```
JAL 0x1234         ! r31 = PC + 4; goto PC + 4 + 0x1234
```


EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-12

05-10

Type R

Fields: Opcode 6, rs1 5, rs2 5, rd 5, func 11.

Used for arithmetic, logical instructions, and moves.

Sometimes just two registers used, but func field needed for operation.

Note that "func" field provides additional coding space.

Examples

```
ADD r1, r2, r3
ADDF f1, f2, f3
MOVI2S f1, r1    ! (rs2 unused)
```


EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.
05-10

05-13

Synthetic Instructions and DLX (NIB)

Misleading (in a nice way) assembly language mnemonics.

Implies a "new" opcode, but really uses an existing one.

Meant for programmer convenience.

Example, set register to zero:

```
CLR r1              ! Synthetic instruction
ADD r1, r0, r0      ! True instruction (DLX)
```

Assembler generates a "ADD r1, r0, r0" when it finds a CLR r1 mnemonic.

Sometimes several true instructions for each synthetic instruction.

EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.

05-13

05-14

Sample Synthetic Instructions (NIB)

No Operation:

```
NOP              ! Synthetic
ADD r0, r0, r0   ! DLX
BNEZ r0, 0       ! DLX
```

Register move:

```
MOVI2I r1, r2    ! Synthetic
ADD r1, r2, r0   ! DLX
```

Bitwise invert:

```
NOT r1, r2       ! Synthetic
XORI r1, r2, #-1 ! DLX
```


EE 4720 Lecture Transparency. Formatted 12:19, 13 February 1998 from lsli05.

05-14