Lecture 7.1 - Rajeev

CS 3810 - Computer Organization
MIPS Instruction Set

What's the purpose of an ISA?

The ISA lies at the HW/SW interface in a computer system.

Let's understand this through a simple example:

```
C       a = b + c;
```

There can be many different types of instructions.

Compiler operations

32-bit M/C instr

a, b, c

hardware-friendly

operands

assembly
32 bits 32 b
4 Bytes 4B

usually

> a line of C will connect to
multiple lines of assembly instructions.

Skip slide # 17 on discussion of IS design
slide # 18 - 21

> There may be different ways
you may generate compiler
assembly code,
some may be better than
others. (use the term
temporary location)

[No discussion on floating point]

\[ f = (q + w) - (i + j) \]

Compiler is
more likely
to generate
the code on
the left (although temporaries are used)
[Due the design of the PL & compiler ... but
it all depends]
Lecture #8

Operand Locations

\[ \text{In x86} \rightarrow 8 \text{ regs.} \\
\text{In MIPS} \rightarrow 32 \text{ regs.} \]

Processor

Register file, containing a set of registers (32)

Each taking 32 bits of size 32

1. In C all vars are located in the memory.

memory (8 GB/16 GB/32 GB)

Explanation

Accessing memory is expensive. To avoid repeatedly accessing memory, move values from memory to registers. In MIPS, in fact, instructions in fact require operands to be register values only.

So, registers are used as a scratchpad.

Note register file is much smaller. Less registers fo more expensive data transfer. More registers means more expensive resources on the processor (real estate)
Register file 32 bit wide

for C/Java vars

32 registers

Temporary variables

64 bit arch are 64 bit wide

Each 32 bit entity stored here is a word.

Question

7 How many addresses can a 32-bit word encode/represent?

\[ 2^{32} = 2^{10} \times 2^{10} \times 2^{10} \times 2 \]

\[ = 1k \times 1k \times 1k \times 4 \]

\[ = 4 \text{ GB billion values} \]

then your address space is 768 GB

7 If you assume each address refers to a 4-byte entity and you have 4 billion unique addresses.
addresses of a
> All 164 GB memory cannot be
accessed by a 32 bit arch.

Accessing The memory

FIRST SHOW SLIDE 24, then switch to 25.2
Simplistic explanation of how the
compiler determines the memory
addresses.

main()
int a, b, c, d[10];

computer creates a
map (var. virt. address)
Each address refers to a
location in a partition of memory.
starts at base (adds)
Each int is 4 bytes
These addresses are in virtual memory.
The computer knows what's where in memory register from the table.

The compiler also knows base address from register $t0 (let's say it's stored there).

Now, say we have:

\[ a = b + c \]

The 1st available register.

Load $b$

\[ \text{load} \ lw \ $s0, 4 \ ($t0) \rightarrow \text{add} \ a \ \text{to address into, memo the value to $s0.} \]

Load $c$

\[ \text{load} \ lw \ $s1, 8 \ ($t0) \]

Add $s2, $s0, $s1 $s2$ corresponds to address $a$.

Store in a 16 byte memory.

\[ \text{sw} \ $s2, 0 ($t0) \]
Immediate Operands

Previously, we had instructions like,

\[
\text{add} \quad \$50, \quad \$51, \quad \$52 \\
\text{addi} \quad \$50, \quad \$51, \quad 1000 \\
\text{Say if want to add 1000 to } \$50 \\
\text{We use, addi } \\
\text{there has to be at least 1 operand as a register}
\]

\[
\text{addi} \quad \$50, \quad \$300, \quad 1000
\]