CS 340
Computer Systems

#4: Instruction Set Architectures (ISAs)

Jan. 26, 2023 · Wade Fagen-Ulmschneider

Instruction Set Architecture (ISA)

Every CPU has a set of commands it understands known as its Instruction Set Architecture or ISA. Two ISAs are **very** common:

1.

2.

An ISA defines the function of the hardware in the CPU – one could say the ISA is the: ____________________.

CPU Registers

CPU Registers are on-dye modules that are physically interconnected with the hardware performing the CPU operations (ex: they're hard-wired to the **ADD** circuit).
...in fact, almost all CPU operations ______________________!

Three key ideas to know about CPU registers:

1. [Size]:

2. [Speed]:

3. [Limited]:

- **x64** (ex: Intel, AMD):
- **ARMv8** (ex: Apple M1/M2, Cell Phones):

CPU Register Names

The 16 general purpose x64 CPU registers have names based on how many bits you're working with:

	64-bits	32-bits	16-bits	8-bits
0	%rax	%eax	%ax	%al
1	%rbx	%ebx	%bx	%bl
2	%rcx	%ecx	%cd	%cl
3	%rdx	%edx	%dx	%dl
4	%rsi	%esi	%si	%sil
5	%rdi	%edi	%di	%dil
6	%dbp	%ebp	%bp	%dpl
7	%rsp	%esp	%sp	%spl
...				

Instruction Sets

Every ISA defines a set of instructions that a CPU can execute:

Move:	MOV, XCHG, PUSH, POP, ...
Arithmetic (int):	ADD, SUB, MUL, DIV, NEG, CMP, ...
Logic:	AND, OR, XOR, SHR, SHL, ...
Control Flow:	JMP, LOOP, CALL, RET, ...
Synchronization:	LOCK, ...
Floating Point:	FADD, FSUB, FMUL, FDIV, FABS, ...

ARM processors have significantly fewer instructions and are known as ____________________________ while x64 processors have a greater set of instructions and known as ______________________.

Q: Advantages of RISC / CISC?

CPU Instruction in a Real Program

04.c

```c
#include <stdio.h>

int main() {
  int a = 0;
  a = a + 3;
  a = a - 2;
  a = a * 4;
  a = a / 2;
  a = a * 5;
  printf("Hi");
  a = a * 479;
  return a;
}
```

To compile a program without optimizations and references back to the original code, the “debug” flag is required:

```
$ gcc -g 04.c
```

Then, we can dump the output object in a human readable format:

```
$ objdump -d ./a.out
```

This result of this command shows **EVERY** operation that the CPU will execute when running the program! The operations that correspond to the main() function are organized to the right (⇒).

One Special Register: ____________________

	04.c	gcc -g 04.c objdump -d ./a.out
3	int main() {	f3 0f 1e fa endbr64 55 push %rbp 48 89 e5 mov %rsp,%rbp 48 83 ec 10 sub $0x10,%rsp
4	int a = 0;	c7 45 fc 00 00 00 00 movl $0x0,-0x4(%rbp)
5	a = a + 3;	83 45 fc 03 addl $0x3,-0x4(%rbp)
6	a = a - 2;	83 6d fc 02 subl $0x2,-0x4(%rbp)
7	a = a * 4;	c1 65 fc 02 shll $0x2,-0x4(%rbp)
8	a = a / 2;	8b 45 fc mov -0x4(%rbp),%eax 89 c2 mov %eax,%edx c1 ea 1f shr $0x1f,%edx 01 d0 add %edx,%eax d1 f8 sar %eax 89 45 fc mov %eax,-0x4(%rbp)
9	a = a * 5;	8b 55 fc mov -0x4(%rbp),%edx 89 d0 mov %edx,%eax c1 e0 02 shl $0x2,%eax 01 d0 add %edx,%eax 89 45 fc mov %eax,-0x4(%rbp)
10	printf("Hi");	48 8d 3d f0 0d 00 00 lea 0xdf0(%rip),%rdi # 2004 <_IO_stdin_used+0x4> b8 00 00 00 00 mov $0x0,%eax e8 42 fe ff ff callq 1060<printf@plt>
11	a = a * 479;	8b 45 fc mov -0x4(%rbp),%eax 69 c0 df 01 00 00 imul $0x1df,%eax,%eax 89 45 fc mov %eax,-0x4(%rbp)

Operation Timings

Q: Do all operations take the same amount of time on the CPU?

Q: What are the CPU timings for various operations?