Lecture 5

18-1-2021

Instruction Set Architecture (ISA)

What is Instruction Set Architecture (ISA)?

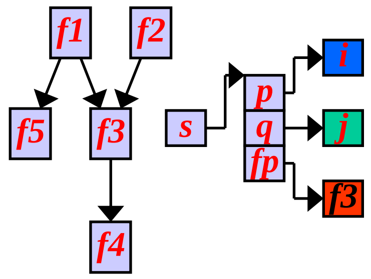
- Programmer visible part of a processor:
  - **Instruction Set** (what operations can be performed (opcodes)?)
  - **Instruction Format** (how are instructions encoded?)
  - **Registers** (where are data located?)
  - ***Addressing Modes*** (how is data loc. specified?)
  - **Exceptional Conditions** (what happens if something goes wrong?)

ISA cont…

- ISA is important:
  - Not only from the programmer's perspective.
  - From processor designer and implementer perspectives as well.

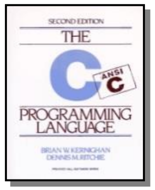
The Big Picture

$$\begin{tikzcd}
\text{Requirements} \arrow[d] \\
\text{Algorithms} \arrow[d] \\
\text{Prog. Lang./OS} \arrow[d] \\
\text{ISA} \arrow[d] \\
\text{uArch} \arrow[d] \\
\text{Gate} \arrow[d] \\
\text{Circuit} \arrow[d] \\
\text{Physical}
\end{tikzcd}$$

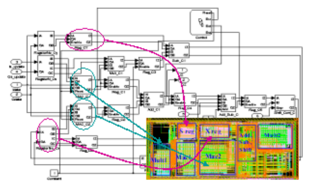


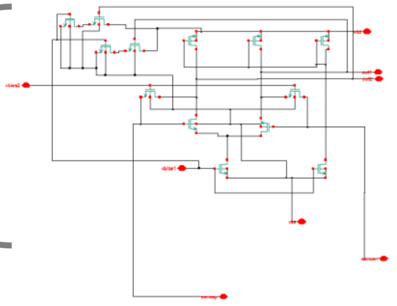
Problem Focus

```
f2() {
  f3(s2, &j, &i);
  *s2->p = 10;
  i = *s2->q + i;
}
```

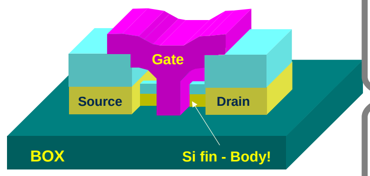


```
i1: ld r1, b          <p1>
i2: ld r2, c          <p1>
i3: ld r5, z          <p3>
i4: mul r6, r5, 3     <p3>
i5: add r3, r1, r2    <p1>
```

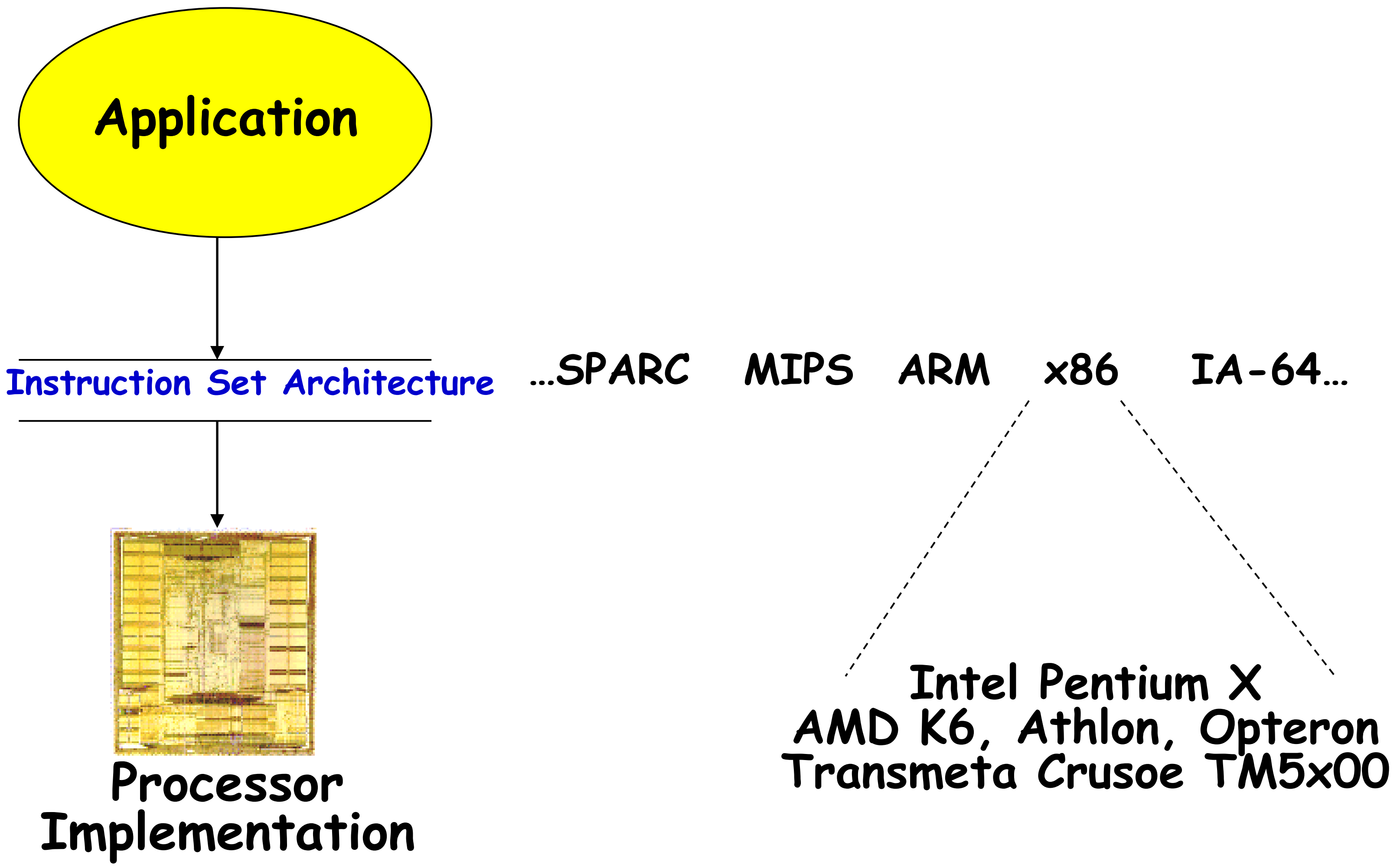




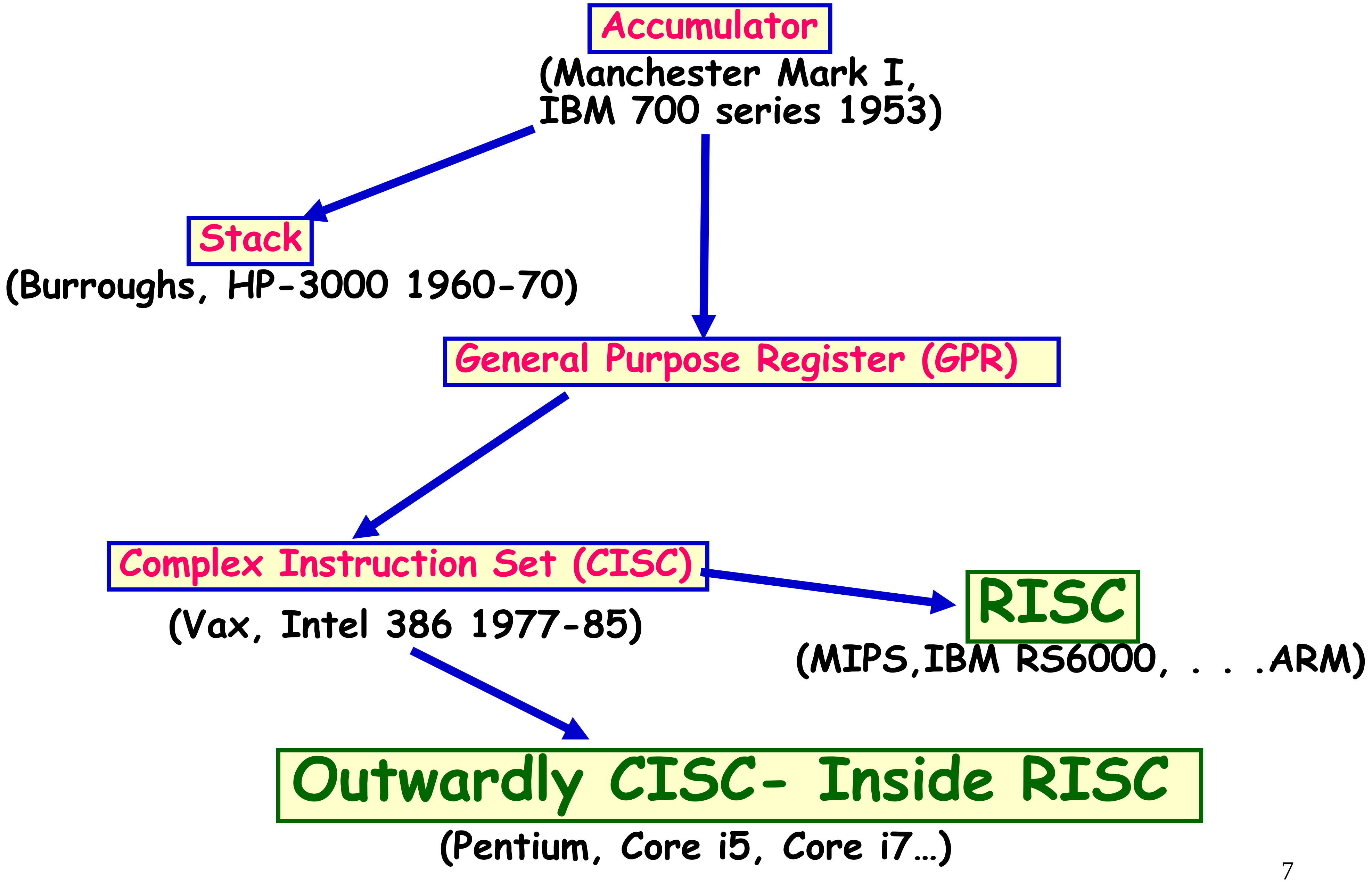
Performance Focus



Instruction Set Architecture

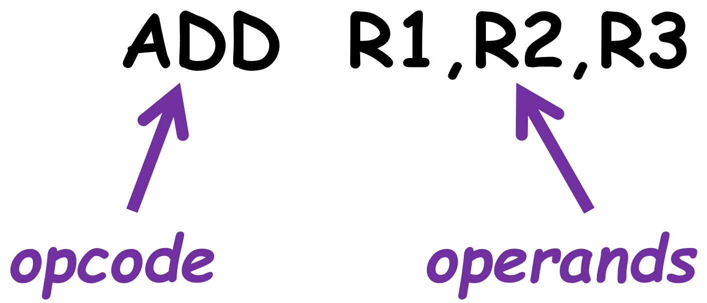


Evolution of Instruction Sets


Accumulator
(Manchester Mark I,
IBM 700 series 1953)
Stack
(Burroughs, HP-3000 1960-70)
General Purpose Register (GPR)
Complex Instruction Set (CISC)
(Vax, Intel 386 1977-85)
RISC
(MIPS,IBM RS6000, . . .ARM)
Outwardly CISC- Inside RISC
(Pentium, Core i5, Core i7…)

Different Types of ISAs

- **Determined by the means used for storing operands in CPU.**



- The major choices are:
  - A stack, an accumulator, or a set of registers.
- What is a Stack architecture?
  - Operands are implicitly on top of the stack.

Accumulator architecture

- One operand is in the accumulator (register) and the others are elsewhere.
  - Popular in older machines... **now dinosaur**...
  - What is the problem?

- General purpose registers:
  - Operands can be in registers or in specific memory locations.

Comparison of Architectures

Consider the instruction: C =A + B

Stack	Accumulator	Register-Memory	Register-Register
Push A	Load A	Load R1, A	Load R1, A
Push B	Add B	Add R1, B	Load R2, B
Add	Store C	Store C, R1	Add R3, R1, R2
Pop C			Store C, R3

Classification of ISAs

Accumulator (before 1960, *e.g.* 68HC11):

1-address	add A	acc ← acc + mem[A]

Stack (1960s to 1970s):

0-address	add	tos ← tos + next

Memory-Memory (1970s to 1980s):

2-address	add A, B	mem[A] ← mem[A] + mem[B]
3-address	add A, B, C	mem[A] ← mem[B] + mem[C]

Register-Memory (1970s to present, *e.g.* core i7):

2-address	add R1, A	R1 ← R1 + mem[A]
	load R1, A	R1 ← mem[A]

Register-Register (Load/Store) (1960s to present, *e.g.* MIPS):

3-address	add R1, R2, R3	R1 ← R2 + R3
	load R1, R2	R1 ← mem[R2]
	store R1, R2	mem[R1] ← R2

Some More Architectural Issues…

- Instruction length needs to be in multiples of bytes. Why?
- Instruction encoding can be:
  - Variable or Fixed
- Variable encoding tries to use as few bits to represent a program as possible:
  - But at the cost of complexity of decoding.
- Fixed Encoding: Alpha, ARM, MIPS

Operation and Modes	**Addr1**	**Addr2**	**Addr3**

What is Meant by Addressing Mode of an Operand?

- Denotes how the location of an operand is specified:
  - Add R1,R2,R3
  - Load R1, 1000
  - Load R1, (R2)

Common Addressing Modes

- **Register address-** Operand is in register
  - Example: **Add R1,R2,R3**
- **Memory Direct Address (aka absolute)**- Operand in memory at given address
  - Example: **Load R1, 1000**
- **Register Indirect Address** - Register contains memory address of operand
  - Example: **Load R1, (R2)**

Common Addressing Modes

- **Indexed Address (Displacement)**- Add base and register value to get address of operand in memory

  Offset

  Base Register

  - Example **Load R1, 100(R2)**

- Used for accessing array elements **X[I]:**
  - Array index **I** is normally in the register and the address field **X** is the first location in the array

Common Addressing Modes

- **Immediate Address** - Uses constant value contained in instruction.
  - **Example** addi R1,R2,4 - adds constant 4 to register R2 and stores result in R1
  - Used when constants required in programming
- **PC relative address** - Address from instruction is added to the current value in the Program Counter
  - **Example** J 4 - Jumps to address PC+4

Exercise

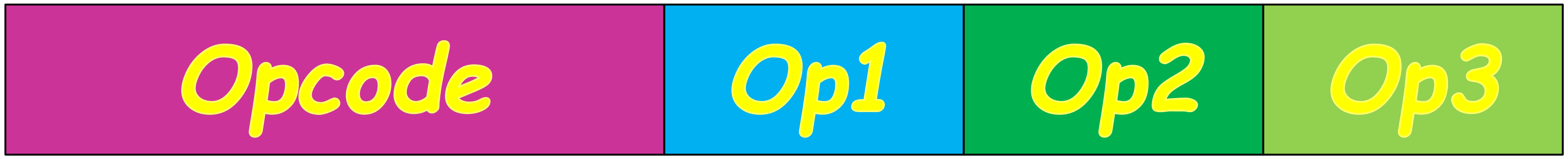
- What addressing modes would you use to write assembly program corresponding to the following code?

```
int i, sum=0;
for(i=0;i<10;i++)
     sum=sum+i;
```

```
        Sub R1,R1,R1
        Mov R3,#10
        Load R2, Sum
Loop:   Add R2, R1,R2
        Add R1,R1,#1
        BNE R1,R3, Loop
```

Encoding an Instruction

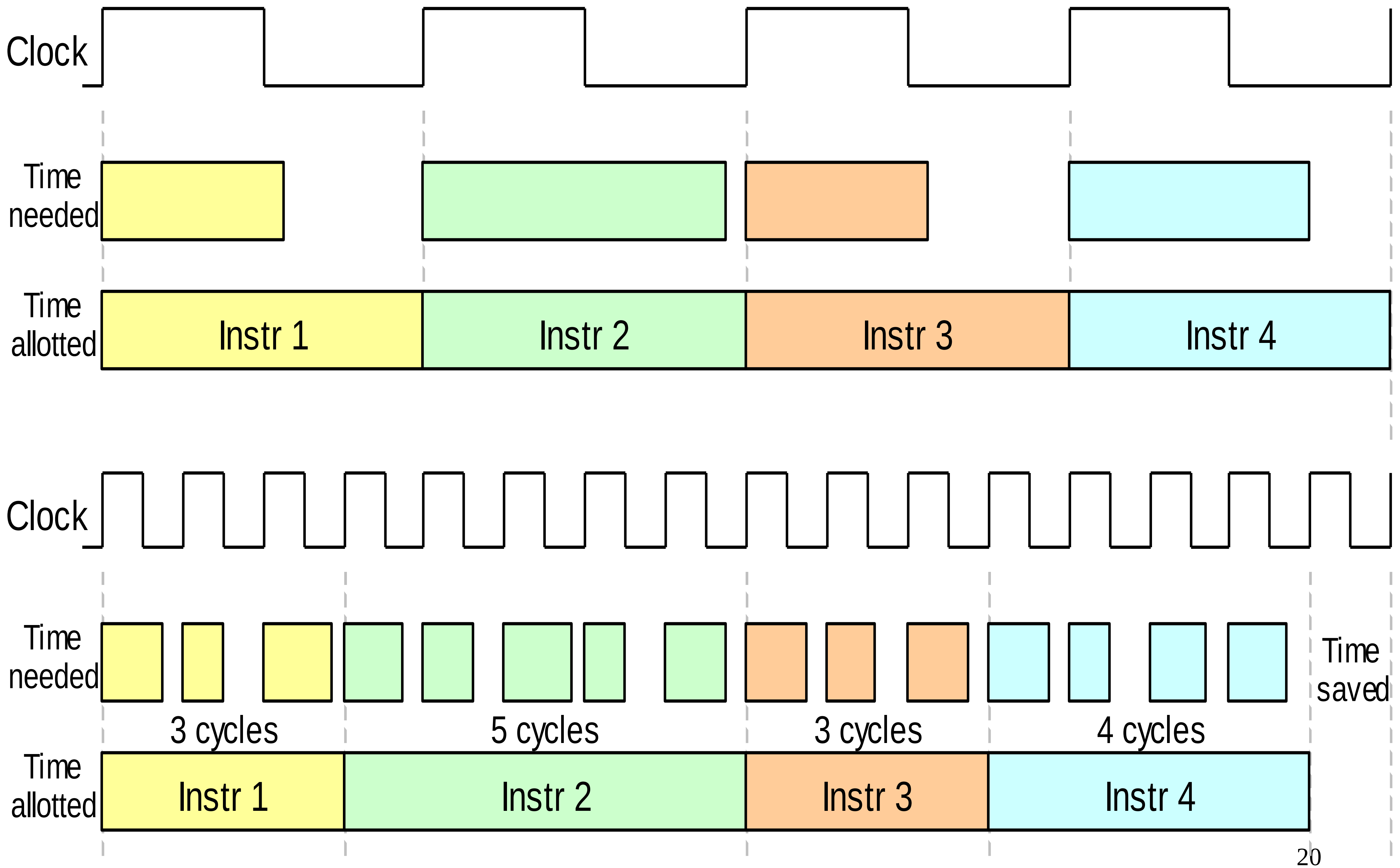
- Consider **ADD R1, R2, R3**
- Size of opcode part of an instruction:
  - Determined by the total number of opcodes supported.
- Each operand needs to encode a register, in register addressing mode



Question

- What is the difference between:
  - A single cycle data path processor and a multi cycle data path processor?
- Which is better?
  - A single cycle processor, or a multi cycle processor?
  - Why?

Single-Cycle vs. Multicycle Datapath



- Single cycle datapath implies the execution of any instruction within one clock cycle
- **Single cycle datapath wastes some time**
- Clock cycle time is determined by the instruction taking the longest time,
  - usually instructions involving memory operations
- In multicycle datapath, different types of instructions may take different cycle times
- **More time efficient, but control circuitry is much more complex.**

RISC-CISC Controversy

Two Radical Instruction Set Architectures

- **CISC**
  - Complex Instruction Set Computer

- **RISC**
  - Reduced Instruction Set Computer

Characteristics of A CISC Processor

- Rich instruction set:
  - Some simple, some very complex
- Complex addressing modes:
  - Orthogonal addressing (Every possible addressing mode for every instruction).
- Large variation in CPI:
  - 1 to 14 cycles for X86
- Instructions are of variable size 1 to12 bytes
- Small number of registers
- Microprogram control
- No (or inefficient) pipelining

CISC Philosophy

- One instruction could do the work of several instructions.
  - Example: A single instruction could load two numbers to be added, add them, and then store the result back to memory directly.

Add C,A,B

- CISC experimental finding:
  - Only about 20% of the available instructions used in a typical program

CISC Philosophy

- Implementing commonly used instructions in hardware holds significant performance benefits.
  - **Example:** Hardware string manipulation instructions could help achieve significant performance improvement…

CISC Philosophy cont…

- Many versions of the same instructions were supported:
  - Different versions did almost the same thing.
- Example:
  - One version would read two numbers from memory, and store the result in a register.
  - Another version would read one number from memory and the other from a register and store the result to memory.

Operand Addressing Seen in CISC…

• Direct (absolute)	ADD R1, (1001)
• Register indirect	SUB R2, (R1)
• Indexed	ADD R1, (R2 + R3)
• Scaled	SUB R2, 100(R2)[R3]
• Autoincrement	ADD R1, (R2)+
• Autodecrement	SUB R2, -(R1)
• Memory indirect	ADD R1, @(R3)

And many more …

Example CISC ISA: Intel X86

12 addressing modes

- Register.
- Immediate.
- Direct.
- Base.
- Base + Displacement.
- Index + Displacement.
- Scaled Index + Displacement.
- Based Index.
- Based Scaled Index.
- Based Index + Displacement.
- Based Scaled Index + Displacement.
- Relative.

Operand sizes:

- Can be 8, 16, 32, 48, 64, or 80 bits long.
- Also supports string operations.

Instruction Encoding:

- The smallest instruction is one byte.
- The longest instruction is 12 bytes long.
- First few bits contain the opcode, mode specifiers, and register fields.
- The remainder bytes are for address displacement and immediate data.

Questions

- How come people suddenly realized in 1984 that RISC is good?

Was CISC a historical blunder?

- Were the people who designed processors before 1984 morons?