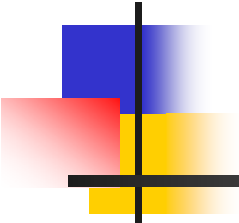


DNA

Microcode and Processor Modes



Alexandre David

1.2.05

adavid@cs.aau.dk





Evolution of Computers

- Early systems
 - CPU (*central* processing unit) controlled the entire system
 - Responsible for I/O, computations, ...
- Modern computers
 - Decentralized architecture
 - Processors distributed (I/O)
 - CPU still controls other processors



General Purpose CPU

- Very complex because
 - designed for wide variety of tasks – multiple roles
 - contains special purpose sub-units
 - ex: core i7 has 731M transistors
 - supports protection and privileges (OS/applic.)
 - supports priorities (I/O)
 - data size (32/64-bit registers)
 - high speed – parallelism = replication



Modes of Execution

- Modes define
 - subset of valid instructions
 - size of data items
 - accessible regions of memory
 - available functional units
 - amount of privilege
- One mode active at any time
 - change with special instructions/registers
 - initiated by hardware
 - mechanisms vary

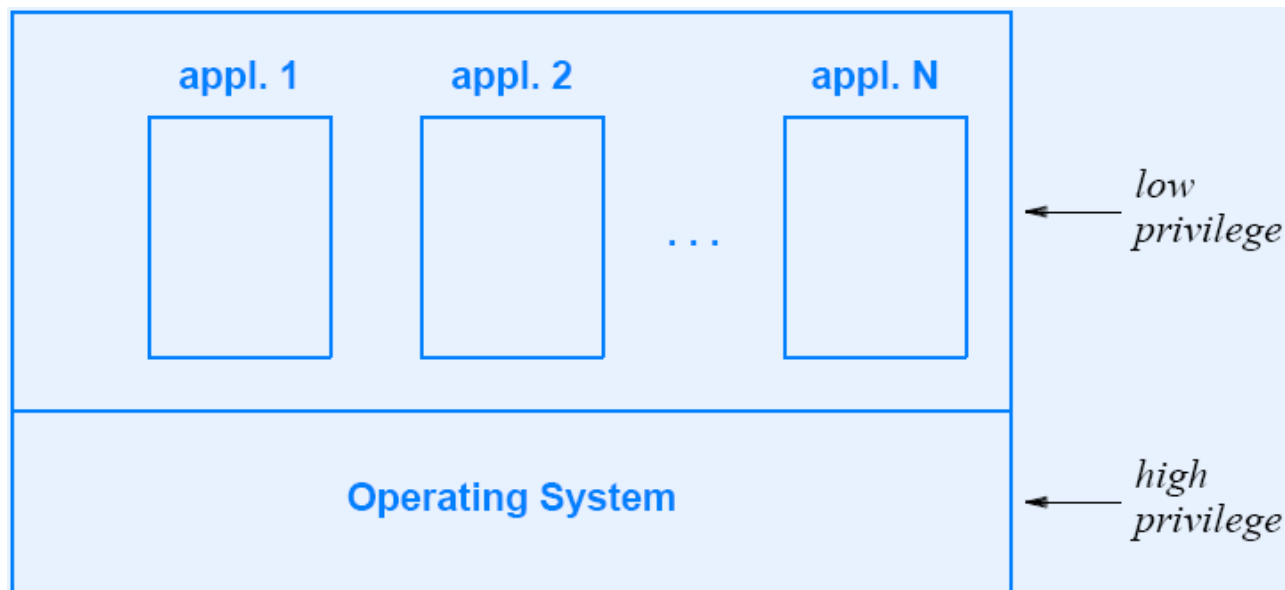


Example: x86

- Real mode
 - 20-bit segmented addressing (1MB)
 - direct access to BIOS
 - no multitasking or memory protection
 - CPUs after 80286 start in real mode
- Protected mode
 - virtual memory, memory protection
 - support for paging
- Virtual 8086 mode
 - run in real mode under 32-bit protected mode
- Long mode
 - 64-bit mode
- Enhanced mode
 - protected mode with SSE instruction

Privilege Level

- Determines available resources.
- Modes define privileges.
- We need at least 2 levels
 - OS
 - Applications

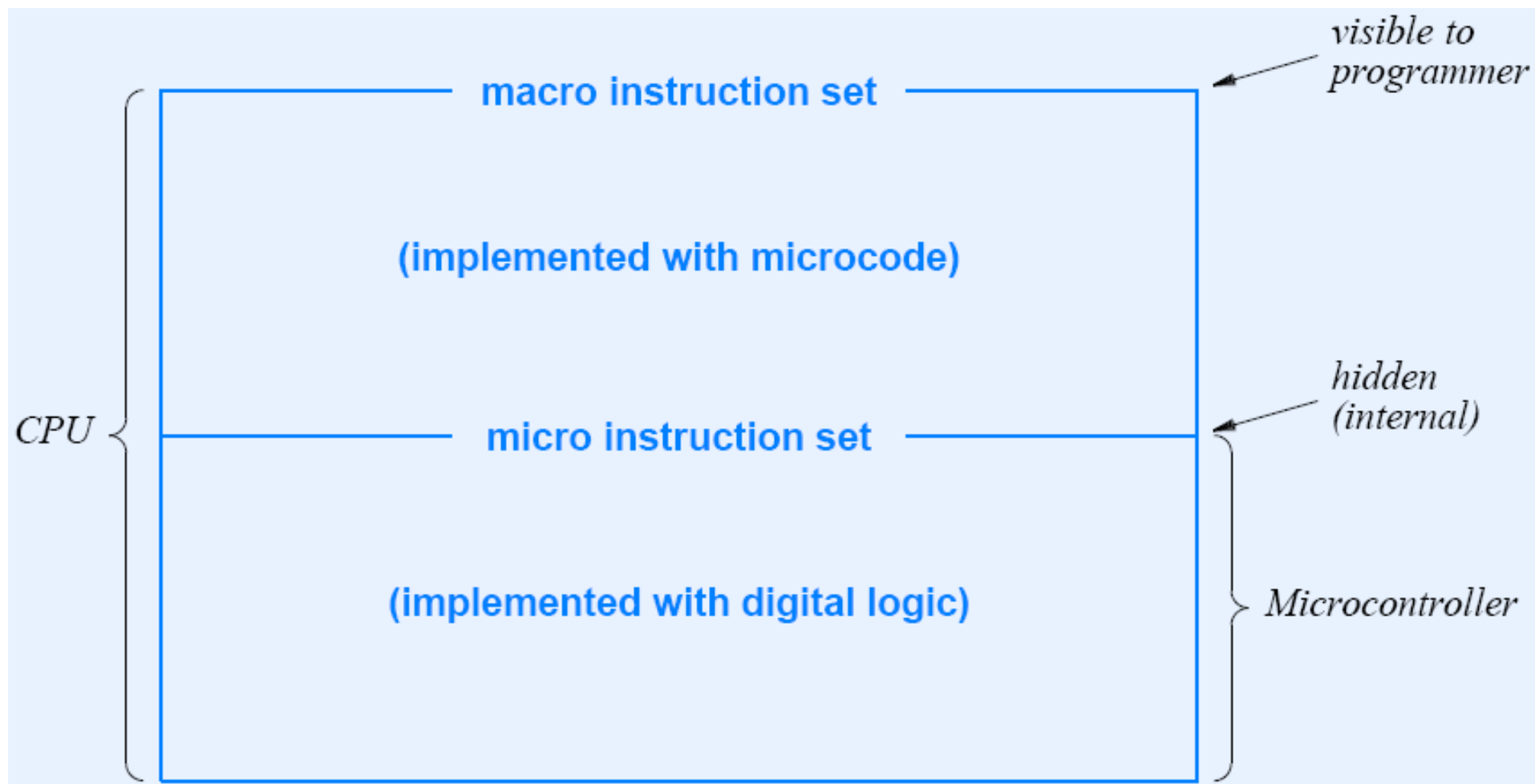




Microcode

- How to implement complex CPU?
 - Program the complex instructions.
 - Visible machine language = macro instruction set.
 - Internal language = micro-code.
 - Microcontroller inside CPUs that decode and execute macro-instructions.
 - RISC
 - Processors are all RISCs in the end.
 - Key: Easier to write programs with micro-code than to build hardware from scratch.

Microcode





Data and Register Sizes

- Size of visible register may differ from size of internal registers.
 - Ex: Could implement 32-bit instruction set on a 16-bit microcontroller.



Example: add32 on 16-bit

add32:

move low-order 16 bits from R5 into r2

move low-order 16 bits from R6 into r3

add r2 and r3, placing result in r1

→ save value of the carry indicator

move high-order 16 bits from R5 into r2

move high-order 16 bits from R6 into r3

add r2 and r3, placing result in r0

copy the value in r0 to r2

→ add r2 and the carry bit, placing the result in r0

→ check for overflow and set the condition code

move the thirty-two bit result from r0 and r1

to the desired destination



Advantages/Drawbacks

- Advantages

- Can change microcode and keep the same macro-instruction set!
- Less prone to errors, can be updated more easily.

- Drawback

- Cost in performance – overhead.
- ~~*Microcontroller must run at high speed than CPU to accommodate multiple micro-instructions per macro-instruction.*~~

Speed=clock defined at the gate level=speed of micro-controller.



Visibility of Microcode

- Fixed microcode – generally the case
- Alterable microcode
 - reconfigurable CPU – not very popular
 - change before running programs – special procedure
 - more flexibility
 - performance overhead
 - FPGAs even more flexible with possible better performance.



Vertical Microcode

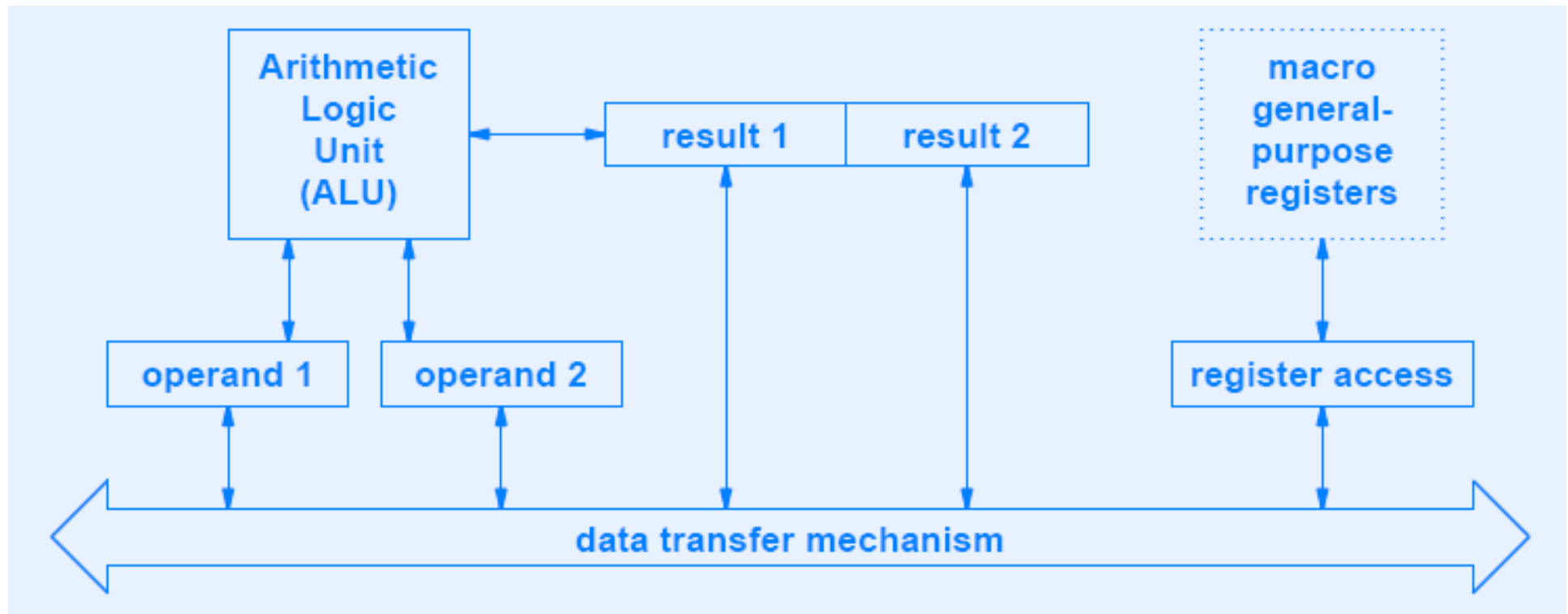
- Simple view of microcontroller ~ standard processor.
- Execution of micro-code like assembly.
- One micro-instruction at a time.
- Access to different units.
- Decode each macro-instruction and execute micro-code.
- Easy to read/write, bad performance.
- Not the case in practice.

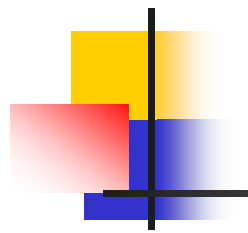


Horizontal Microcode

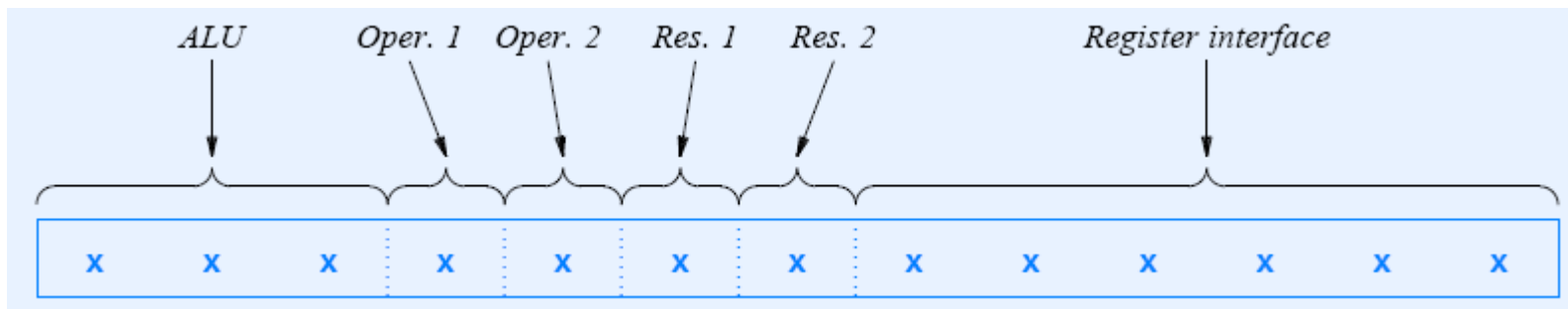
- Use implicit parallelism.
 - Utilize units in parallel when possible.
- Control data movements and the different hardware units *at the same time*.
- Very difficult to program.
- Long instruction:
|exec op1 unit1|exec op2 unit2|transfer this register there|...

Example Architecture



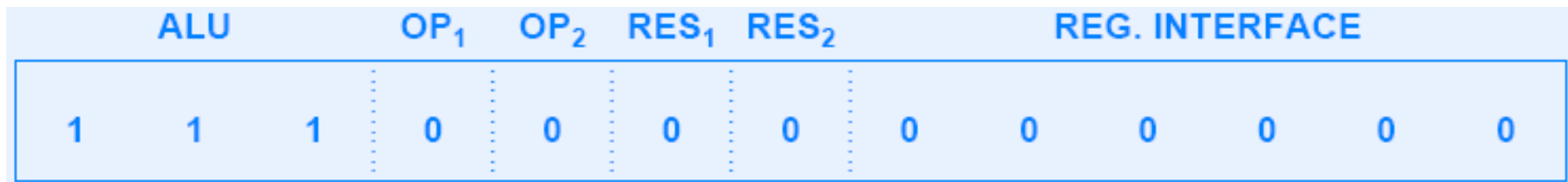


Unit	Command	Meaning
ALU	0 0 0	No operation
	0 0 1	Add
	0 1 0	Subtract
	0 1 1	Multiply
	1 0 0	Divide
	1 0 1	Left shift
	1 1 0	Right shift
	1 1 1	Continue previous operation
operand 1 or 2	0	No operation
	1	Load value from data transfer mechanism
result 1 or 2	0	No operation
	1	Send value to data transfer mechanism
register interface	0 0 x x x x	No operation
	0 1 x x x x	Move register xxxx to data transfer mechanism
	1 0 x x x x	Move data transfer mechanism to register xxxx
	1 1 x x x x	No operation



Horizontal Microcode

- Not like conventional programs.
- Each instruction takes one cycle
 - but not all operations take one cycle
 - special care for timing, wait for units that need more cycles



continue



Intelligent Microcontroller

- Schedules instructions & units.
- Handles operations in parallel.
- Performs branch prediction.
 - May try 2 paths and discard the results of the wrong one later.
 - Important: Keep the sequential semantics.
- Out-of-order execution
 - use scoreboard to keep track of results and dependencies



Conclusion

- Does it matter?
 - Yes! Understand your hardware and its technology.
 - Use it in a better way. Reduce branches, or make them easy to guess.

Ex:

```
for(i = 0, j = n-1; i < j; ++i, --j) swap(&a[i], &a[j])
```

harder than

```
for(i = 0; i < n/2; ++i) swap(&a[i], &a[n-1-i])
```