CSAIL

Instruction Set Evolution in the Sixties: *GPR, Stack, and Load-Store Architectures*

Arvind
Computer Science and Artificial Intelligence Laboratory
M.I.T.

Based on the material prepared by Arvind and Krste Asanovic

The Sixties

- Hardware costs started dropping
  - memories beyond 32K words seemed likely
  - separate I/O processors
  - large register files
- Systems software development became essential
  - Operating Systems
  - I/O facilities
- Separation of Programming Model from implementation become essential
  - family of computers

CSAIL

Issues for Architects in the Sixties

- Stable base for software development
- Support for operating systems
  - processes, multiple users, I/O
- Implementation of high-level languages
  - recursion, ...
- Impact of large memories on instruction size
- How to organize the *processor state* from the programming point of view
- Architectures for which fast implementations could be developed

Three Different Directions in the Sixties

- A machine with only a high-level language interface
  - Burrough's 5000, a stack machine
- A family of computers based on a common ISA
  - IBM 360, a General Register Machine
- A pipelined machine with a fast clock (Supercomputer)
  - CDC 6600, a Load/Store architecture

The Burrough's B5000:
An ALGOL Machine, Robert Barton, 1960

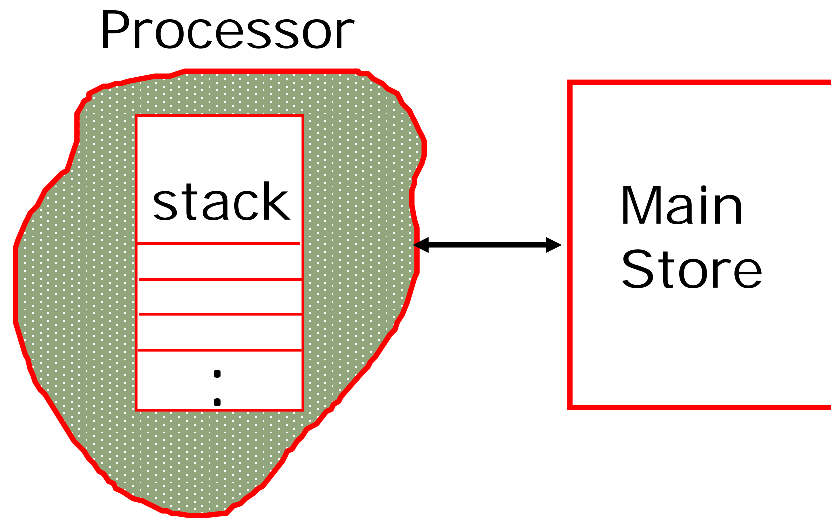
- Machine implementation can be completely hidden if the programmer is provided only a high-level language interface.

- *Stack machine* organization because stacks are convenient for:
  1. expression evaluation;
  2. subroutine calls, recursion, nested interrupts;
  3. accessing variables in block-structured languages.

- B6700, a later model, had many more innovative features
  - tagged data
  - virtual memory
  - multiple processors and memories

CSAIL

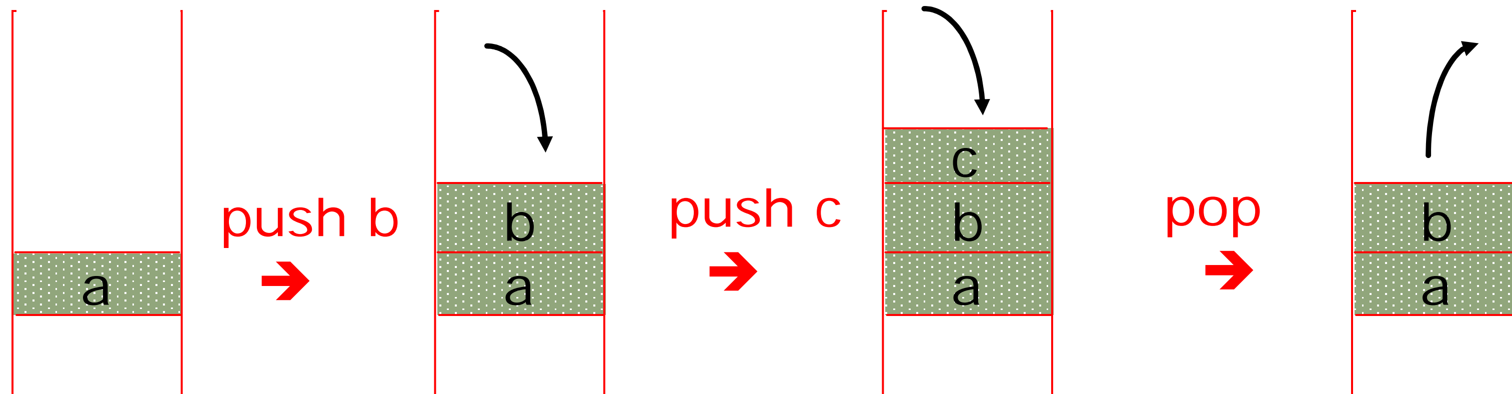
A Stack Machine



A Stack machine has a stack as a part of the processor state

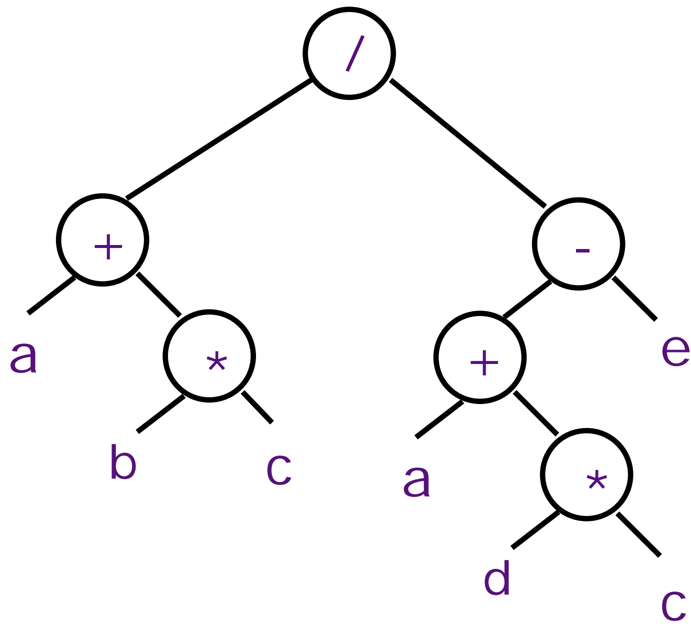
typical operations:
push, pop, +, *, ...

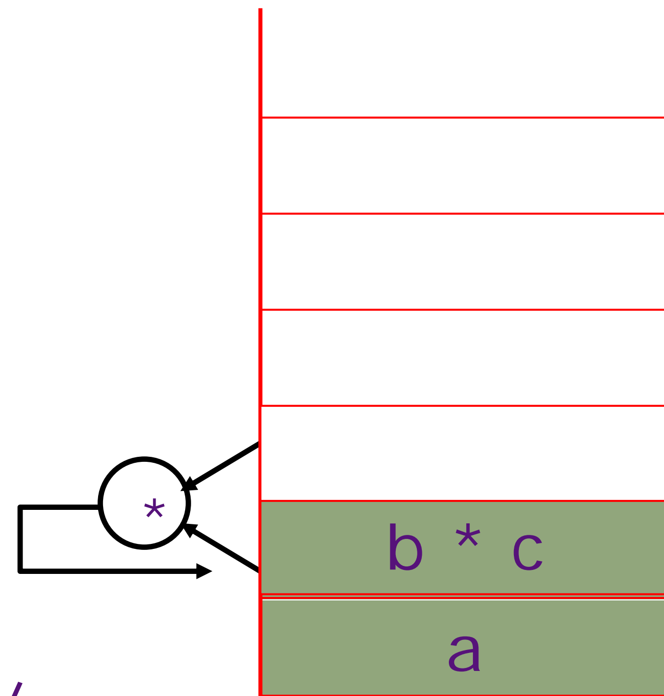
Instructions like + implicitly specify the top 2 elements of the stack as operands.



Evaluation of Expressions

(a + b * c) / (a + d * c - e)





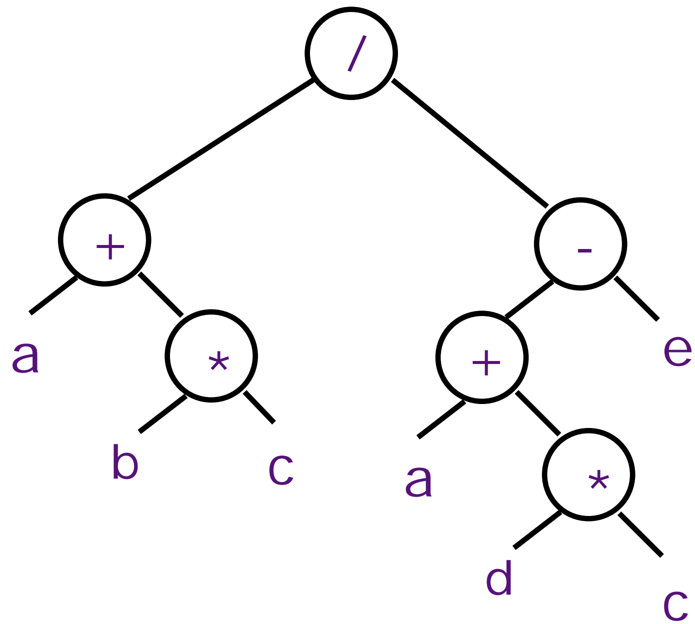
Reverse Polish

a b c * + a d c * + e - /

push a Push c
Push b
multiply

Evaluation of Expressions

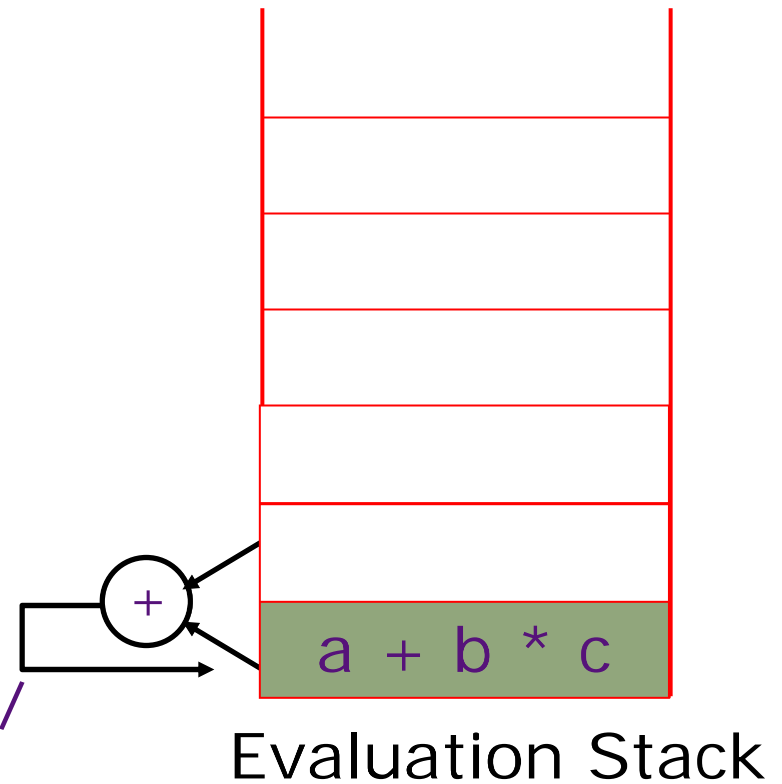
(a + b * c) / (a + d * c - e)



Reverse Polish

a b c * + a d c * + e - /

add



Evaluation Stack

Hardware organization of the stack

- Stack is part of the processor state
  - ⇒ *stack must be bounded and small*
    - ≈ number of Registers, *not* the size of main memory
- Conceptually stack is unbounded
  - ⇒ *a part of the stack is included in the processor state; the rest is kept in the main memory*

Stack Size and Memory References

a b c * + a d c * + e - /

program	*stack (size = 2)*	*memory refs*
push a	R0	a
push b	R0 R1	b
push c	R0 R1 R2	c, ss(a)
*	R0 R1	sf(a)
+	R0	
push a	R0 R1	a
push d	R0 R1 R2	d, ss(a+b*c)
push c	R0 R1 R2 R3	c, ss(a)
*	R0 R1 R2	sf(a)
+	R0 R1	sf(a+b*c)
push e	R0 R1 R2	e,ss(a+b*c)
-	R0 R1	sf(a+b*c)
/	R0	

4 stores, 4 fetches (implicit)

CSAIL

Stack Operations and Implicit Memory References

- Suppose the top 2 elements of the stack are kept in registers and the rest is kept in the memory.

Each *push* operation	$\Rightarrow$	1 memory reference
pop operation	$\Rightarrow$	1 memory reference

No Good!

- Better performance can be got if the top N elements are kept in registers and memory references are made only when register stack overflows or underflows.

Issue - when to Load/Unload registers ?

CSAIL

Stack Size and Expression Evaluation

a b c * + a d c * + e - /

a and c are "loaded" twice

. ⇒
not the best use of registers!

program	*stack (size = 2)*
push a	R0
push b	R0 R1
push c	R0 R1 R2
*	R0 R1
+	R0
push a	R0 R1
push d	R0 R1 R2
push c	R0 R1 R2 R3
*	R0 R1 R2
+	R0 R1
push e	R0 R1 R2
-	R0 R1
/	R0

CSAIL

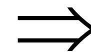
Register Usage in a GPR Machine

(a + b * c) / (a + d * c - e)

	Load	R0	a
	Load	R1	c
	Load	R2	b
Reuse R2	Mul	R2	R1
	Add	R2	R0
	Load	R3	d
Reuse R3	Mul	R3	R1
	Add	R3	R0
Reuse R0	Load	R0	e
	Sub	R3	R0
	Div	R2	R3

More control over register usage since registers can be named explicitly

Load	Ri m
Load	Ri (Rj)
Load	Ri (Rj) (Rk)

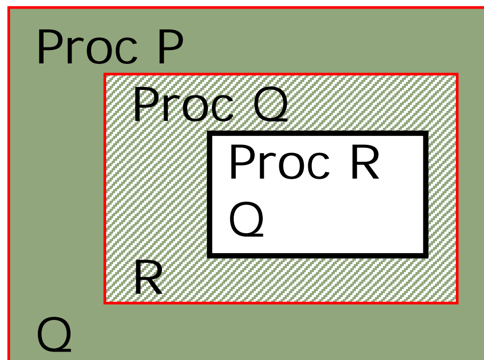


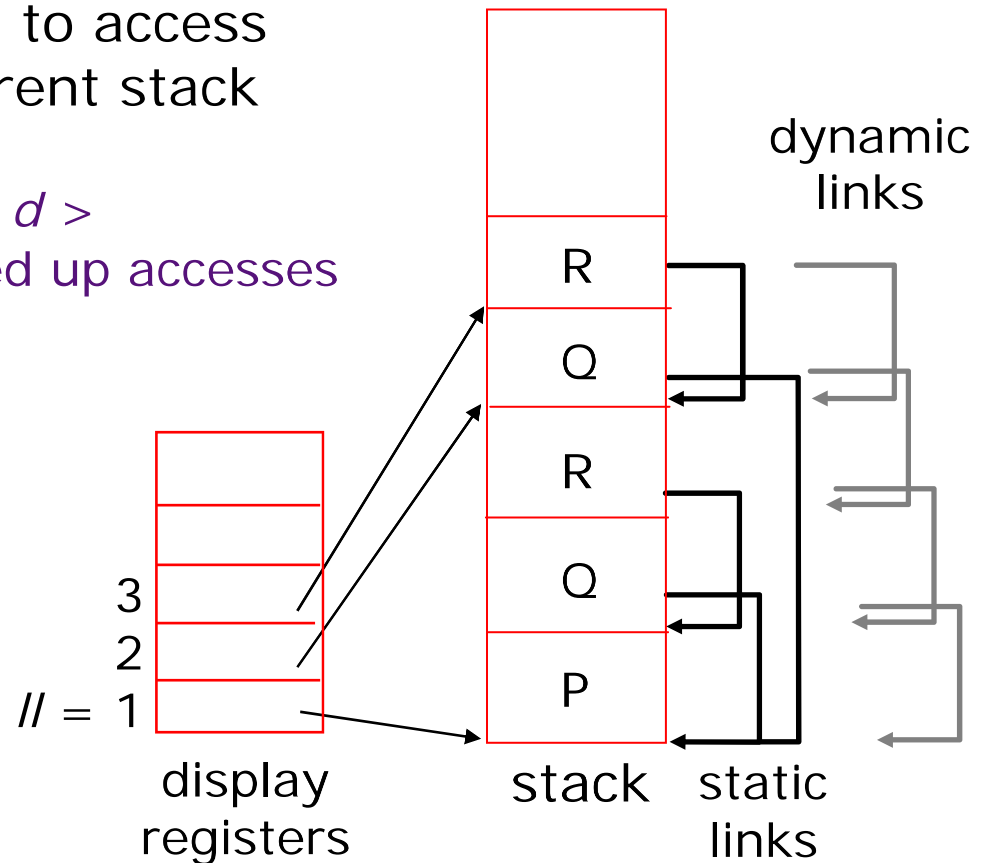
- *eliminates unnecessary Loads and Stores*
- *fewer Registers*

but instructions may be longer!

Procedure Calls

- Storage for procedure calls also follows a stack discipline
- However, there is a need to access variables beyond the current stack frame
  - lexical addressing < *ll* , *d* >
  - display registers to speed up accesses to stack frames

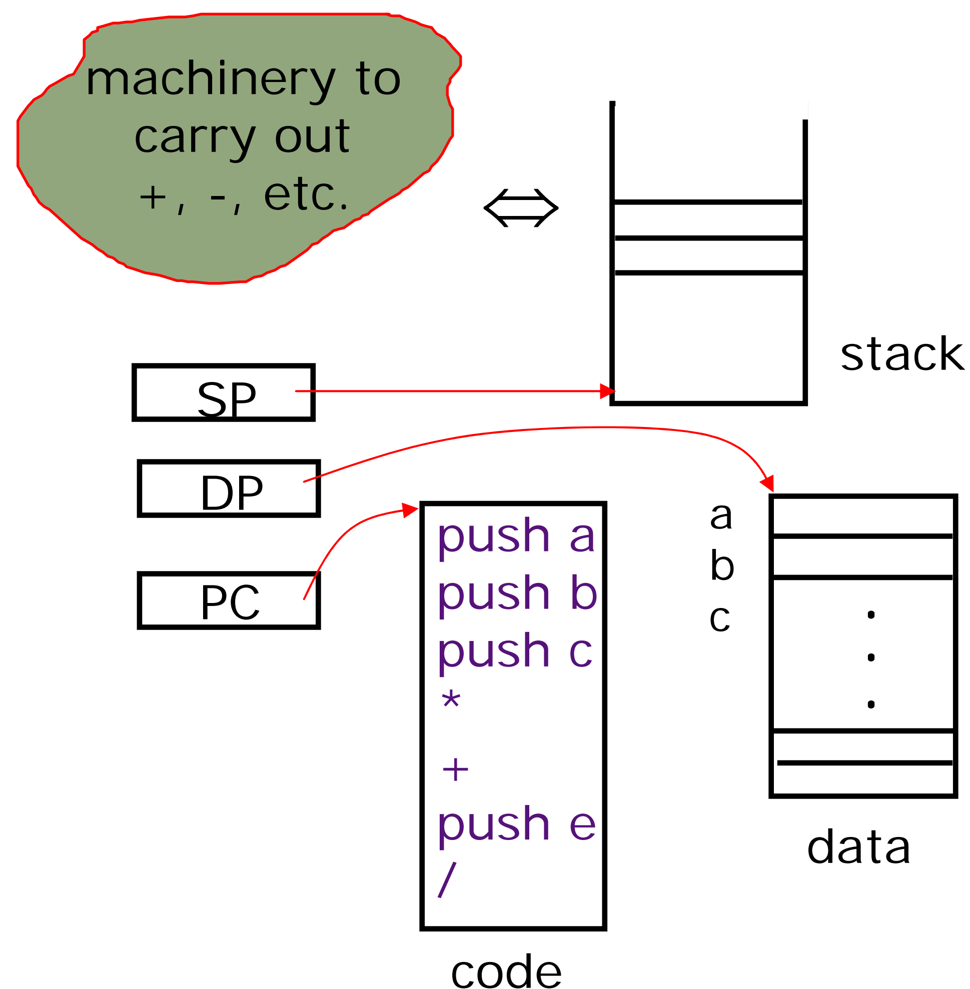




automatic loading of display registers?

Stack Machines: Essential features

- In addition to push, pop, + etc., the instruction set must provide the capability to
  - *refer to any element in the data area*
  - *jump to any instruction in the code area*
  - *move any element in the stack frame to the top*



Stack versus GPR Organization

Amdahl, Blaauw and Brooks, 1964

1. The performance advantage of push down stack organization is derived from the presence of fast registers and not the way they are used.
2. "Surfacing" of data in stack which are "profitable" is approximately 50% because of constants and common subexpressions.
3. Advantage of instruction density because of implicit addresses is equaled if short addresses to specify registers are allowed.
4. Management of finite depth stack causes complexity.
5. Recursive subroutine advantage can be realized only with the help of an independent stack for addressing.
6. Fitting variable length fields into fixed width word is awkward.

CSAIL

Stack Machines (Mostly) Died by 1980

1. Stack programs are not smaller if short (Register) addresses are permitted.

2. Modern compilers can manage fast register space better than the stack discipline.

3. Lexical addressing is a useful abstract model for compilers but hardware support for it (i.e. display) is not necessary.

GPR's and caches are better than stack and displays

Early language-directed architectures often did not take into account the role of compilers!

B5000, B6700, HP 3000, ICL 2900, Symbolics 3600

Stacks post-1980

- Inmos Transputers (1985-2000)
  - Designed to support many parallel processes in Occam language
  - Fixed-height stack design simplified implementation
  - Stack trashed on context swap (fast context switches)
  - Inmos T800 was world's fastest microprocessor in late 80's
- Forth machines
  - Direct support for Forth execution in small embedded real-time environments
  - Several manufacturers (Rockwell, Patriot Scientific)
- Java Virtual Machine
  - Designed for software emulation not direct hardware execution
  - Sun PicoJava implementation + others
- Intel x87 floating-point unit
  - Severely broken stack model for FP arithmetic
  - Deprecated in Pentium-4 replaced with SSE2 FP registers

CSAIL

A five-minute break to stretch your legs

IBM 360: A General-Purpose Register (GPR) Machine

- Processor State
  - 16 General-Purpose 32-bit Registers
    - *may be used as index and base register*
    - *Register 0 has some special properties*
  - 4 Floating Point 64-bit Registers
  - A Program Status Word (PSW)
    - *PC, Condition codes, Control flags*
- A 32-bit machine with 24-bit addresses
  - No instruction contains a 24-bit address !
- Data Formats
  - 8-bit bytes, 16-bit half-words, 32-bit words, 64-bit double-words

IBM 360: Precise Interrupts

- IBM 360 ISA (Instruction Set Architecture) preserves sequential execution model
- Programmers view of machine was that each instruction either completed or signaled a fault before next instruction began execution
- Exception/interrupt behavior constant across family of implementations

CSAIL

IBM 360: Original family

	Model 30	. . .	*Model 70*
Storage	8K - 64 KB		256K - 512 KB
Datapath	8-bit		64-bit
Circuit Delay	30 nsec/level		5 nsec/level
Local Store	Main Store		Transistor Registers
Control Store	Read only 1μsec		Conventional circuits

IBM 360 instruction set architecture completely hid the underlying technological differences between various models.

With minor modifications it survives till today

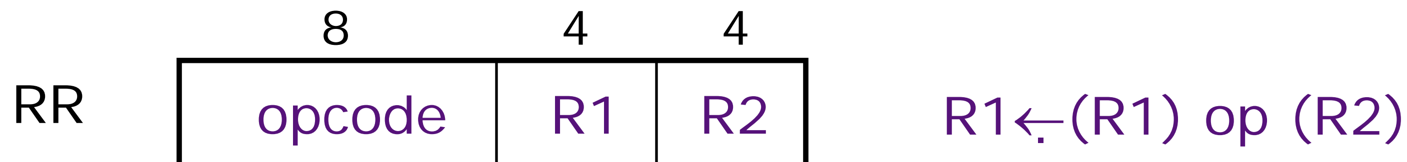
CSAIL

IBM S/390 z900 Microprocessor

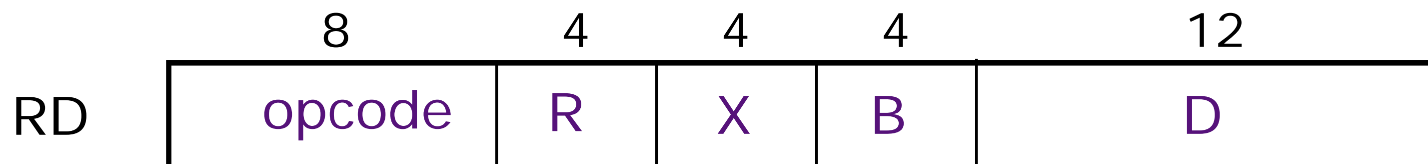
- 64-bit virtual addressing
  - first 64-bit S/390 design (original S/360 was 24-bit, and S/370 was 31-bit extension)
- 1.1 GHz clock rate (announced ISSCC 2001)
  - 0.18μm CMOS, 7 layers copper wiring
  - 770MHz systems shipped in 2000
- Single-issue 7-stage CISC pipeline
- Redundant datapaths
  - every instruction performed in two parallel datapaths and results compared
- 256KB L1 I-cache, 256KB L1 D-cache on-chip
- 20 CPUs + 32MB L2 cache per Multi-Chip Module
- Water cooled to 10°C junction temp

CSAIL

IBM 360: Some Addressing Modes



R1←(R1) op (R2)



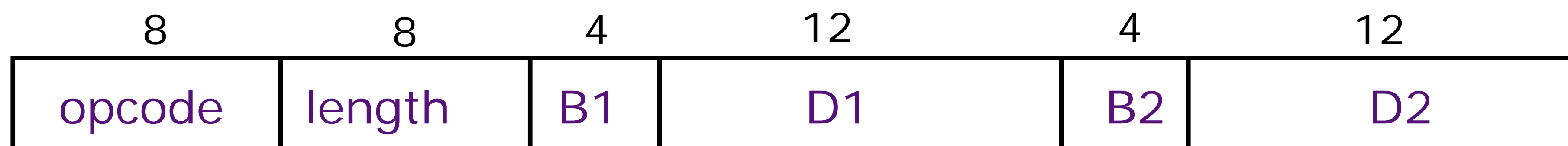
R ← (R) op M[(X) + (B) + D]

a 24-bit address is formed by adding the 12-bit displacement (D) to a base register (B) and an Index register (X), if desired

The most common formats for arithmetic & logic instructions, as well as Load and Store instructions

CSAIL

IBM 360: Character String Operations



SS format: store to store instructions

M[(B1) + D1] ← M[(B1) + D1] op M[(B2) + D2]

iterate "length" times

Most operations on decimal and character strings use this format

MVC move characters
MP multiply two packed decimal strings
CLC compare two character strings
...

Multiple memory operations per instruction complicates exception & interrupt handling

CSAIL

IBM 360: Branches & Condition Codes

- Arithmetic and logic instructions set *condition codes*
  - equal to zero
  - greater than zero
  - overflow
  - carry...
- I/O instructions also set condition codes
  - channel busy
- Conditional branch instructions are based on testing condition code registers (CC's)
  - RX and RR formats
    - BC_ branch conditionally
    - BAL_ branch and link, i.e., R15 ← (PC)+1 *for subroutine calls*

      ⇒ CC's must be part of the PSW

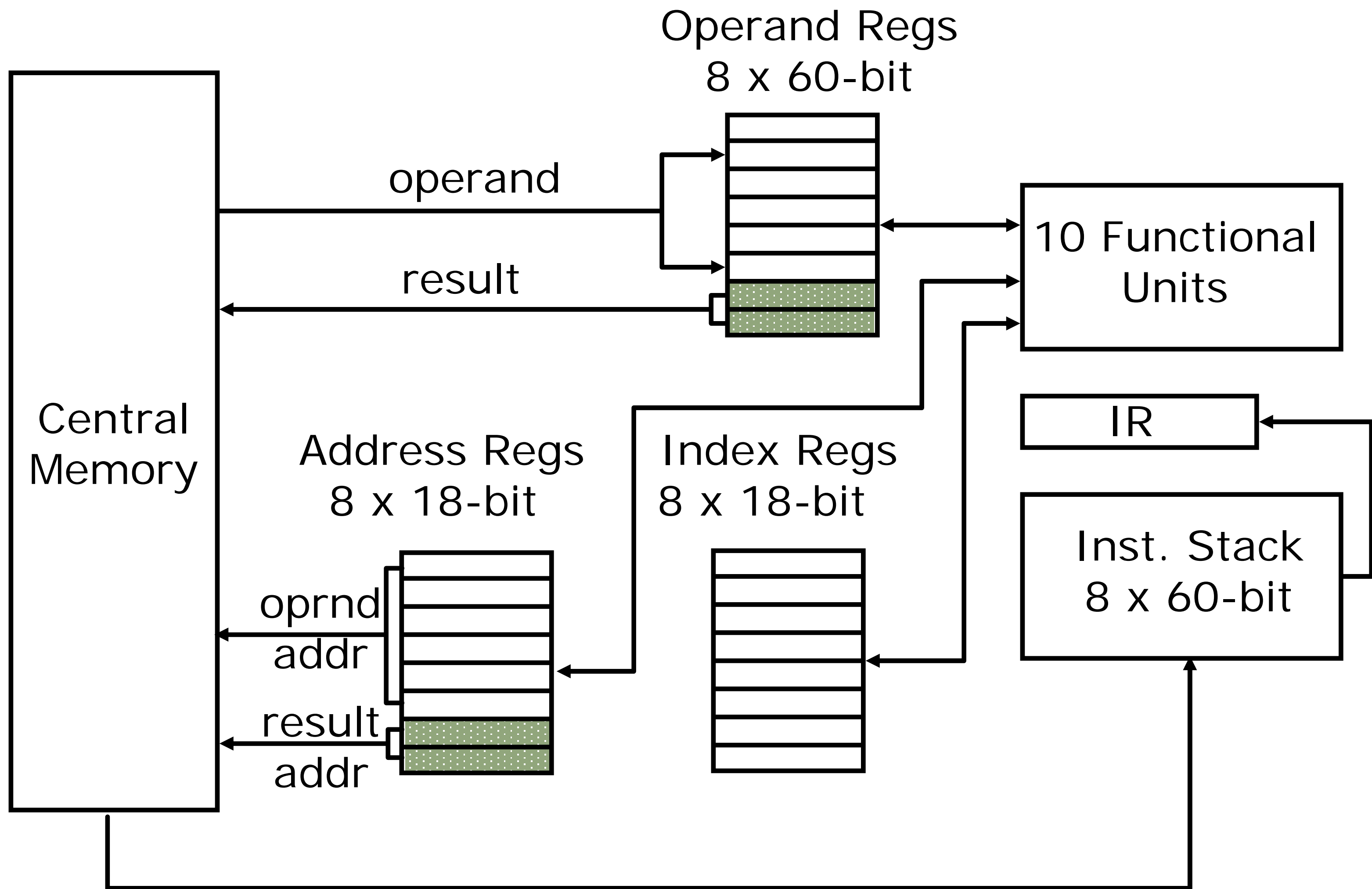
CSAIL

CDC 6600 *Seymour Cray, 1964*

- A fast pipelined machine with 60-bit words
- Ten functional units
  - Floating Point: adder, multiplier, divider
  - Integer: adder, multiplier

  ...
- Hardwired control (no microcoding)
- Dynamic scheduling of instructions using a scoreboard
- Ten Peripheral Processors for Input/Output
  - a fast time-shared 12-bit integer ALU
- Very fast clock
- Novel freon-based technology for cooling

CDC 6600: Datapath



CDC 6600: A Load/Store Architecture

- Separate instructions to manipulate three types of reg.
  - 8 60-bit data registers (X)
  - 8 18-bit address registers (A)
  - 8 18-bit index registers (B)

- All arithmetic and logic instructions are reg-to-reg

Ri ← (Rj) op (Rk)

- Only Load and Store instructions refer to memory!

Ri ← M[(Rj) + disp]

Touching address registers 1 to 5 initiates a load
6 to 7 initiates a store
- *very useful for vector operations*

CDC6600: Vector Addition

```
        B0 ← - n
loop:   JZE  B0, exit
        A0 ← B0 + a0      load X0
        A1 ← B0 + b0      load X1
        X6 ← X0 + X1
        A6 ← B0 + c0      store X6
        B0 ← B0 + 1
        jump loop
```

Ai = address register
Bi = index register
Xi = data register

CSAIL

What makes a good instruction set?

One that provides a simple software interface yet allows simple, fast, efficient hardware implementations

... but across 25+ year time frame

Example of difficulties:

- Current machines have register files with more storage than entire main memory of early machines!
- On-chip test circuitry in current machines has hundreds of times more transistors than entire early computers!

CSAIL

Full Employment for Architects

- Good news: "Ideal" instruction set changes continually
  - Technology allows larger CPUs over time
  - Technology constraints change (e.g., now it is power)
  - Compiler technology improves (e.g., register allocation)
  - Programming styles change (assembly, HLL, object-oriented, ...)
  - Applications change (e.g., multimedia, ...)
- Bad news: Software compatibility imposes huge damping coefficient on instruction set innovation
  - Software investment dwarfs hardware investment
  - Innovate at microarchitecture level, below the ISA level (this is what most computer architects do)
- New instruction set can only be justified by new large market and technological advantage
  - Network processors
  - Multimedia processors
  - DSP's