Microprocessor Evolution - from 4-bit Ones to Superscalar RISC
Pavel Píša, Michal Štepanovský

Czech Technical University in Prague, Faculty of Electrical Engineering
## Early Technology and Complexity Comparison

<table>
<thead>
<tr>
<th>CPU</th>
<th>Company</th>
<th>Year</th>
<th>Transis.</th>
<th>Technology</th>
<th>Reg/Bus</th>
<th>Data/prog+IO</th>
<th>Cache</th>
<th>Float</th>
<th>Frequency</th>
<th>MIPS</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>4004</td>
<td>Intel</td>
<td>1971</td>
<td>2,300</td>
<td>10um - 3x4mm</td>
<td>4bit</td>
<td>1kB/4kB</td>
<td></td>
<td></td>
<td>750kHz</td>
<td>0.06</td>
<td>$200</td>
</tr>
<tr>
<td>8008</td>
<td>Intel</td>
<td>1972</td>
<td>3,500</td>
<td>10um</td>
<td>8bit</td>
<td>16kB</td>
<td></td>
<td></td>
<td>0.06</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8080</td>
<td>Intel</td>
<td>1974</td>
<td>6,000</td>
<td>6um</td>
<td>8bit</td>
<td>64kB+256</td>
<td></td>
<td></td>
<td>2MHz</td>
<td>0.64</td>
<td>$150</td>
</tr>
<tr>
<td>MC6501</td>
<td>NMOS T.</td>
<td>1975</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$20</td>
<td></td>
</tr>
<tr>
<td>8085</td>
<td>Intel</td>
<td>1976</td>
<td>6,500</td>
<td>3um</td>
<td>8bit</td>
<td>64kB+256</td>
<td></td>
<td></td>
<td>5MHz</td>
<td>0.37</td>
<td></td>
</tr>
<tr>
<td>Z-80</td>
<td>Zilog</td>
<td>1976</td>
<td></td>
<td></td>
<td>8bit</td>
<td>64kB+256</td>
<td></td>
<td></td>
<td>2.5MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MC6502</td>
<td>NMOS T.</td>
<td>1976</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$25</td>
<td></td>
</tr>
<tr>
<td>8086</td>
<td>Intel</td>
<td>1978</td>
<td>29,000</td>
<td>3um</td>
<td>16/16bit</td>
<td>1MB+64kB</td>
<td></td>
<td></td>
<td>4.77MHz</td>
<td>0.33</td>
<td>$360</td>
</tr>
<tr>
<td>8088</td>
<td>Intel</td>
<td>1979</td>
<td></td>
<td></td>
<td>16/8bit</td>
<td>1MB+64kB</td>
<td></td>
<td></td>
<td>4.77MHz</td>
<td>0.33</td>
<td></td>
</tr>
<tr>
<td>MC68000</td>
<td>Motorola</td>
<td>1979</td>
<td>68,000</td>
<td></td>
<td>16-32/16bit</td>
<td>16MB</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80286</td>
<td>Intel</td>
<td>1982</td>
<td>134,000</td>
<td>1.5um</td>
<td>16/16bit</td>
<td>16MB/1GBvirt</td>
<td>256B/0B</td>
<td></td>
<td>6MHz</td>
<td>0.9</td>
<td>$380</td>
</tr>
<tr>
<td>MC68020</td>
<td>Motorola</td>
<td>1984</td>
<td>190,000</td>
<td></td>
<td>32/32bit</td>
<td>16MB</td>
<td>Ano</td>
<td></td>
<td>16MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>80386DX</td>
<td>Intel</td>
<td>1985</td>
<td>275,000</td>
<td>1.5um</td>
<td>32/32bit</td>
<td>4GB/64TBvirt</td>
<td></td>
<td></td>
<td>16MHz</td>
<td></td>
<td>$299</td>
</tr>
<tr>
<td>MC68030</td>
<td>Motorola</td>
<td>1987</td>
<td>273,000</td>
<td></td>
<td>4GB+MMU</td>
<td>256B/256B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80486</td>
<td>Intel</td>
<td>1989</td>
<td>1.2mil</td>
<td>1um</td>
<td>32/32bit</td>
<td>4GB/64TBvirt</td>
<td>8kB</td>
<td>Ano</td>
<td>25MHz</td>
<td>20</td>
<td>$900</td>
</tr>
<tr>
<td>MC68040</td>
<td>Motorola</td>
<td>1989</td>
<td>1.2mil</td>
<td></td>
<td></td>
<td>4GB+MMU</td>
<td>4kB/4kB</td>
<td>Ano</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PowerPC 601</td>
<td>Mot.+IBM</td>
<td>1992</td>
<td>2.8mil</td>
<td></td>
<td>32/64bit</td>
<td>256</td>
<td>32kB</td>
<td>Ano</td>
<td>66MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PA-RISC</td>
<td>HP</td>
<td>1992</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>50MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pentium</td>
<td>Intel</td>
<td>1993</td>
<td>3.1mil</td>
<td>0.8um - BiCMOS</td>
<td>32/64bit</td>
<td>4GB+MMU</td>
<td>Ano</td>
<td></td>
<td>66MHz</td>
<td>112</td>
<td></td>
</tr>
<tr>
<td>Alpha</td>
<td>DEC</td>
<td>1994</td>
<td>9.3mil</td>
<td></td>
<td>64bit</td>
<td>4GB/64TBvirt</td>
<td>8/8+96kB</td>
<td></td>
<td>300MHz</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>MC68060</td>
<td>Motorola</td>
<td>1994</td>
<td>2.5mil</td>
<td></td>
<td></td>
<td>4GB+MMU</td>
<td>8kB/8kB</td>
<td>Ano</td>
<td>50MHz</td>
<td>100</td>
<td>$308</td>
</tr>
<tr>
<td>Pentium Pro</td>
<td>Intel</td>
<td>1995</td>
<td>5.5mil</td>
<td></td>
<td></td>
<td>4GB+MMU</td>
<td>Ano</td>
<td></td>
<td>200/60MHz</td>
<td>440</td>
<td>$1682</td>
</tr>
<tr>
<td>Pentium II</td>
<td>Intel</td>
<td>1998</td>
<td>7.5mil</td>
<td></td>
<td>32/64bit</td>
<td></td>
<td>Ano+MMX</td>
<td></td>
<td>400/100MHz</td>
<td>832</td>
<td></td>
</tr>
<tr>
<td>PowerPC G4MPC7400</td>
<td>Motorola</td>
<td>1999</td>
<td>0.15um - cooper6LM CMOS</td>
<td>64/128bit</td>
<td>4GB/252</td>
<td>32kB/32kB +2MB</td>
<td>Ano+AV</td>
<td></td>
<td>450MHz</td>
<td>825</td>
<td></td>
</tr>
</tbody>
</table>
Accumulator Based Architectures

- register + accumulator → accumulator
  - 4-bit Intel 4004 (1971)
  - 8-bit Intel 8080 (1974) – registers pairs used to address data in 64kB address space
  - basic arithmetic-logic operations only – addition, subtraction, rotation for 8-bit accumulator
  - subroutines by CALL and RET instructions with 16-bit PC save on stack
  - a few 16-bit operations – increment/decrement of registers pairs, addition to HL and save to stack
  - microcode controlled/microprogrammed instructions execution – 2 to 11 clock cycles per instruction at 2 MHz clock signal
Intel 8080

Fast memory ⇒ reduce register count and add address modes

- Motorola 6800, NMOS T. 6502 (1975) - accumulator, index, SP a PC only – use zero page as fast data, CU hardwired

- Texas TMS990 – workspace pointer only, even PC, SP, other registers in main memory, similar to transputers
Memory is bottleneck now ⇒ complex instruction set modeled according to C language constructs, CISC

• Intel 8086 (16-bit upgrade to 8080)
  • 8 × 8-bit register form 4 pairs (16-bit registers), additional 4 16-bit registers – SP, BP (C functions frame), SI (source-index), DI (destination index), 1 MB address space by segments, register+=register, memory+=register, register+=memory

• Motorola 68000 (1979) – 16/32bit
  • two operand instructions
  • register+=register, memory+=register, register+=memory, even one instruction memory=memory
  • based on microcode to process so rich instruction set

• Z-8000 16bit, Z-80000 32bit (1986) CISC
  • 6 phases pipelined execution, without microcode, 18000 transistors only
## Intel 8086 and 32-bit i386

### General-Purpose Registers

<table>
<thead>
<tr>
<th>Segment</th>
<th>31</th>
<th>16</th>
<th>15</th>
<th>8</th>
<th>7</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulator</td>
<td></td>
<td></td>
<td></td>
<td>AH</td>
<td>AL</td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td></td>
<td></td>
<td></td>
<td>BH</td>
<td>BL</td>
<td></td>
</tr>
<tr>
<td>Count</td>
<td></td>
<td></td>
<td></td>
<td>CH</td>
<td>CL</td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td>DH</td>
<td>DL</td>
<td></td>
</tr>
<tr>
<td>Source Index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SI</td>
</tr>
<tr>
<td>Destination Index</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>DI</td>
</tr>
<tr>
<td>Base Pointer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>BP</td>
</tr>
<tr>
<td>Stack Pointer</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SP</td>
</tr>
</tbody>
</table>

### Segment Registers

<table>
<thead>
<tr>
<th>Segment</th>
<th>15</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Segment</td>
<td>CS</td>
<td></td>
</tr>
<tr>
<td>Data Segment</td>
<td>DS</td>
<td></td>
</tr>
<tr>
<td>Stack Segment</td>
<td>SS</td>
<td></td>
</tr>
<tr>
<td>Extra Segment</td>
<td>ES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FS</td>
<td></td>
</tr>
<tr>
<td></td>
<td>GS</td>
<td></td>
</tr>
</tbody>
</table>

### Program Status and Control Register

- **EFLAGS**
  - **EIP**
  - **ESP**
  - **ESI**
  - **EDI**
  - **EBP**
  - **ESP**

**16-bit**
- AX
- BX
- CX
- DX
- SI
- DI
- BP
- SP

**32-bit**
- EAX
- EBX
- ECX
- EDX
- ESI
- EDI
- EBP
- ESP
### Basic Integer Registers of M68xxx/CPU32/ColdFire

#### User mode

<table>
<thead>
<tr>
<th>Register</th>
<th>Bits</th>
<th>Description</th>
<th>Bit Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>D0</td>
<td>7</td>
<td>DATA REGISTERS</td>
<td>0</td>
</tr>
<tr>
<td>D1</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D7</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A0</td>
<td>7</td>
<td>ADDRESS REGISTERS</td>
<td>0</td>
</tr>
<tr>
<td>A1</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>A7 (USP)</td>
<td>7</td>
<td>USER STACK POINTER</td>
<td>0</td>
</tr>
<tr>
<td>PC</td>
<td>32</td>
<td>PROGRAM COUNTER</td>
<td>0</td>
</tr>
<tr>
<td>CCR</td>
<td>32</td>
<td>CONDITION CODE REGISTER</td>
<td>0</td>
</tr>
</tbody>
</table>

#### System mode

<table>
<thead>
<tr>
<th>Register</th>
<th>Bits</th>
<th>Description</th>
<th>Bit Positions</th>
</tr>
</thead>
<tbody>
<tr>
<td>A7# (SSP)</td>
<td>7</td>
<td>SUPERVISOR STACK POINTER</td>
<td>0</td>
</tr>
<tr>
<td>SR</td>
<td>7</td>
<td>STATUS REGISTER</td>
<td></td>
</tr>
<tr>
<td>VBR</td>
<td>7</td>
<td>VECTOR BASE REGISTER</td>
<td></td>
</tr>
<tr>
<td>SFC</td>
<td>7</td>
<td>ALTERNATE FUNCTION</td>
<td></td>
</tr>
<tr>
<td>DFC</td>
<td>7</td>
<td>CODE REGISTERS</td>
<td></td>
</tr>
</tbody>
</table>
Status Register – Conditional Code Part

- **N** – negative ... = 1 when the most significant bit of the result is set; otherwise cleared. (the result is negative for two's complement representation)
- **Z** – zero ... = 1 when result is zero – all bits are zero
- **V** – overflow .. = 1 when an arithmetic overflow occurs implying that the result cannot be represented in the operand size (signed case for add, sub, …)
- **C** – carry ... = 1 when when a carry out of the most significant bit occurs (add) or a borrow occurs (sub)
- **X** - extend (extended carry) .. Set to the value of the C-bit for arithmetic operations; otherwise not affected or set to a specified result
Status Register – System Byte

- **T1, T0** – trace … if some of these bits is set then exception is generated after every instruction execution or when program flow changes (jump, call, return)
- **S** – supervisor … if set to 1 then CPU runs in the supervisor state/mode and SP maps to SSP. Else CPU runs in user mode, SP maps to USP and changes to the system byte are not possible and user mode privileges rules/restrictions are applied to memory access (controlled by MMU).
- **I2, I1, I0** - interrupt mask … up to this interrupt priority level are requests blocked/masked – i.e. they need to wait. The level 7 is exception because it is non-maskable, i.e. exception acceptance cannot be delayed.
Addressing Modes – Basic 68000 Modes

- **Up to 14 addressing modes for operand selection**
- **Rn** operand represents value of data Dn or address An register
- **(An)** memory content at the address specified by An
- **(An)+** memory content at An with following An increment by value equivalent to the operand length (post-increment)
- -(An) the An register is decremented by operand size first and then specifies memory located operand (pre-decrement)
- **(d16,An)** memory at An + 16-bit sign extended offset
- **(d8,An,Xn)** memory at An + 8-bit sign extended offset + index register (another Am or Dm) which can be eventually limited to lower 16 bits, index can be multiplied by 1, 2, 4 or 8 for CPU32 and 68020+ processors
- **(xxx).W** 16-bit absolute address – upper and lower 32kB
- **(xxx).L** 32-bit absolute address
Data throughput and instruction fetching slow still ⇒ cache memory

- The problem has been solved quite well
- Common cache or Harvard arrangement I & D
- More levels (speed limited for bigger size – decoder, capacitance of common signals)
- But requires to solve data coherence when DMA access or SMP is used
  - synchronization instructions for peripherals access and synchronization eieio (PowerPC), mcr p15 (ARM), …
  - hardware support required for caches and SMP
    - protocol MSI, MESI (Pentium), MOSI
    - MOESI AMD64 (Modified, Owned, Exclusive, Shared, and Invalid)
Data Coherence and Multiple Cached Access

MOESI protocol

- Modified – cache line contains actual and modified data, none of other CPUs works with data, old/previous data are hold in main memory
- Owned – line holds actual data, line can be shared with other CPUs CPU but only in S state, main memory is not required to be up to date
- Exclusive – only this CPU and main memory contains cache line data
- Shared – cache line is shared with other CPUs, one of them can be in O state, then data can differ to content in main memory
- Invalid – cache line does not hold any valid data

http://en.wikipedia.org/wiki/MOESI_protocol

<table>
<thead>
<tr>
<th></th>
<th>M</th>
<th>O</th>
<th>E</th>
<th>S</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>O</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>E</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>S</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>I</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Yet faster instructions execution ⇒ RISC architectures

• Reduce data flow dependency between instructions, three operand instructions, speculative instructions execution, more registers to reduce memory accesses, register renaming, eliminate interdependencies on conditional code/flag register (MIPS RISC-V eliminate CC altogether, DEC Alpha, multiple flag registers PowerPC, flags update suppress ARM)

• load-store architecture, computation only register+=register and or register=register+register and separate load-store instructions.

• Fixed instruction encoding ⇒ programs are usually longer but much faster instructions decoding, optimized for pipelined execution
Pipelined execution, branches cause even more problems

- Early branch and jump instructions decode
- Processes instructions in delay slots MIPS, DSP
- Static and dynamic branch prediction, branch target address buffer (cache), speculative instructions execution

Source: wikipedia
Parallel Instructions Execution – Superscalar CPU

In order

Issue

Execute

Finish

Out of order

Reservation stations

Reorder / Completion buffer

In order

Complete

Store buffer

Retire

Fetch

Instruction / decode buffer

Decode

Dispatch buffer

Dispatch

Store buffer
Other techniques to reduce memory access frequency ⇒
register windows, link/return address register

- Generally more registers (RISCs usually 31+1, ARM 32-bit 16, compare with 32-bit 386 still only 8 registers)
- SPARC - 8 global registers, 8 from previous window (parameters), 16 in actual window, up to 100 and more registers to stack windows. 8 registers in actual window is used to pass parameters into subroutine
- PowerPC, MIPS, ARM – speedup to call leaf-node functions with use of return address (link register) to store address of the instruction to be executed after return from subroutine
PowerPC Architecture

USER MODEL VEA

FPR0
FPR1

Condition Register

CR

GPR0
GPR1

Floating-Point Status and Control Register

FPSCR

GPR31

USER MODEL UISA

FPR0
FPR1

Machine State Register

MSR

GPR0
GPR1

Supervisor-Level SPRs

CR

GPR31

USER MODEL OEA

FPR0
FPR1

Development Support SPRs

FPR31

GPR0
GPR1

Memory Management Registers

GPR31

Integer Exception Reg. (XER0)

Link Register (LR)

Count Register (CTR)

Tim. B. Lower - Read (TBL)

Tim. B. Upper - Read (TBU)
Summit Supercomputer – IBM AC922 – 2018 TOP500 #1

- June 2018, US Oak Ridge National Laboratory (ORNL), 200 PetaFLOPS, 4600 “nodes”, 2× IBM Power9 CPU +
  - 6× Nvidia Volta GV100
  - 96 lanes of PCIe 4.0, 400Gb/s
  - NVLink 2.0, 100GB/s CPU2GPU
  - GPU-to-GPU
  - 2TB DDR4-2666 per node
  - 1.6 TB NV RAM per node
  - 250 PB storage
  - POWER9-SO, Global Foundries 14nm FinFET, 8×10⁹ tran., 17-layer, 24 cores, 96 threads (SMT4)
  - 120MB L3 eDRAM (2 CPU 10MB), 256GB/sv

Source: http://www.tomshardware.com/
SPARC – Register Windows

- CPU includes from 40 to 520 general purpose 32-bit registers
- 8 of them are global registers, remaining registers are divided in groups of 16 into at least 2 (max 32) register windows
- Each instruction has access to 8 global registers and 24 registers accessible through actually selected register windows position
- 24 windowed registers are divided into 8 input (in), 8 local (local) and 8 registers from the following window which are visible through current window as an output (out) registers (registers to prepare call arguments)
- Active window is given by value of 5-bit pointer – Current Window Pointer (CWP).
- CWP is decremented when subroutine is entered which selects following window as an active/current one
- Increment of CWP return to the previous register window
- Window Invalid Mask (WIM) is a bit-map which allows to mark any of windows as invalid and request exception (overflow or underflow) when window is activated/selected by CWP
SPARC - Registers

<table>
<thead>
<tr>
<th>R31</th>
<th>Return from actual window ... %i7</th>
</tr>
</thead>
<tbody>
<tr>
<td>R30</td>
<td>The frame pointer %fp ... %i6</td>
</tr>
<tr>
<td>R29</td>
<td>%i5</td>
</tr>
<tr>
<td>R28</td>
<td>%i4</td>
</tr>
<tr>
<td>R27</td>
<td>%i3</td>
</tr>
<tr>
<td>R26</td>
<td>%i2</td>
</tr>
<tr>
<td>R25</td>
<td>%i1</td>
</tr>
<tr>
<td>R24</td>
<td>%i0</td>
</tr>
<tr>
<td>R23</td>
<td>%i7</td>
</tr>
<tr>
<td>R22</td>
<td>%i6</td>
</tr>
<tr>
<td>R21</td>
<td>%i5</td>
</tr>
<tr>
<td>R20</td>
<td>%i4</td>
</tr>
<tr>
<td>R19</td>
<td>%i3</td>
</tr>
<tr>
<td>R18</td>
<td>%i2</td>
</tr>
<tr>
<td>R17</td>
<td>%i1</td>
</tr>
<tr>
<td>R16</td>
<td>%i0</td>
</tr>
<tr>
<td>R15</td>
<td>CALL out return address ... %o7</td>
</tr>
<tr>
<td>R14</td>
<td>The stack pointer %sp ... %o6</td>
</tr>
<tr>
<td>R13</td>
<td>%o5</td>
</tr>
<tr>
<td>R12</td>
<td>%o4</td>
</tr>
<tr>
<td>R11</td>
<td>%o3</td>
</tr>
<tr>
<td>R10</td>
<td>%o2</td>
</tr>
<tr>
<td>R9</td>
<td>%o1</td>
</tr>
<tr>
<td>R8</td>
<td>%o0</td>
</tr>
<tr>
<td>%g7</td>
<td>used by system %g1</td>
</tr>
<tr>
<td>%g6</td>
<td>zero %g0</td>
</tr>
<tr>
<td>%g5</td>
<td></td>
</tr>
<tr>
<td>%g4</td>
<td></td>
</tr>
<tr>
<td>%g3</td>
<td></td>
</tr>
<tr>
<td>%g2</td>
<td></td>
</tr>
</tbody>
</table>

I (in)
L (local)
O (out)
SPARC – Register Windows Operation

CWP=0 (current window pointer)

CANRESTORE=1
w7 ins
w6 outs
w6 local
w5 ins
w5 outs
W5 locals
w4 ins
w4 outs
w3 ins
w3 outs
w3 locals
w2 ins
w2 outs
w2 local
w1 ins
w1 locals
w1 outs
w0 ins
w0 outs
w0 locals
RESTORE SAVE

OTHERWIN=2
w7 ins
w6 ins
w5 ins
W5 locals

CANSAVE=3
w6 ins
w5 ins
w4 ins
w3 ins
w2 ins
w1 ins
w0 ins
w7 ins
w7 locals
w1 outs
w2 outs
w3 outs
w4 outs
w5 outs
w6 outs
SPARC in Space – Cobham Gaisler GR740

- Fault-tolerant, quad-core, SPARC V8, 7-stage pipeline, 8 register windows, 4x4 KiB I + 4x4 KiB D cache, IEEE-754, 2 MiB L2 cache

Source: https://www.gaisler.com/index.php/products/components/gr740
Quad-Core LEON4FT (GR740) Development Board

MIPS Architecture Variants

- Probably architecture with the highest use count on Earth at one time (all kinds of AP, embedded systems, etc.)
- Development still continues even for high performance desktops and supercomputers use – Loongson3A
- MIPS Aptiv – MIPS32 MCU for embedded applications
- MIPS Warrior – MIPS P6600 MIPS64 Release 6 – hardware virtualization with hardware table walk, 128-bit SIMD
- MIPS architecture inspired many SoftCore designs for FPGA, examples
  - Xilinx Microblaze
  - Altera Nios
Pipelined execution, no microcode, but still problems with jump instructions

- Early jump instruction decode
- Use delay slots to keep pipeline busy, MIPS, DSP
- Static and dynamic conditional branch prediction, branch target address cache, speculative instruction execution
MIPS 64-bit Base of China Computing

- Loongson 3A5000 – desktop processor, 12nm, 4 cores @ 2.5GHz
- Loongson 3C5000 – server processor, 12nm, 16 cores, supports 4 to 16-way servers
Attempts to enhance code density ⇒ shorter aliases, variable instruction length even for RISC, VLIW

- ARM, 16bit aliases for most common 32bit instructions (Thumb mode, requires mode switching, later on function by function call basis)
- MIPS Aptiv, same on function basis
- M-Core, 32-bit CPU but only 16-bit instruction encoding
- SuperH, 32/64-bit CPU, 16-bit instructions encoding
- ColdFire - RISC implementation based on 68000 instruction set, but only 16, 32, 48-bit length instructions are accepted
- RISC-V, reserved bits of 32-bit instructions to allow seamless combination of 32 and 16-bit instructions
ARM Architecture - Registers

Current Visible Registers

Abort Mode

- r0
- r1
- r2
- r3
- r4
- r5
- r6
- r7
- r8
- r9
- r10
- r11
- r12
- r13 (sp)
- r14 (lr)
- r15 (pc)
- cpsr
- spsr

Banked out Registers

<table>
<thead>
<tr>
<th></th>
<th>User</th>
<th>FIQ</th>
<th>IRQ</th>
<th>SVC</th>
<th>Undef</th>
</tr>
</thead>
<tbody>
<tr>
<td>r0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r13 (sp)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>r14 (lr)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

spsr
Register, optionally with shift operation

- Shift value can be either be:
  - 5 bit unsigned integer
  - Specified in bottom byte of another register.
- Used for multiplication by constant

Immediate value

- 8 bit number, with a range of 0-255.
  - Rotated right through even number of positions
- Allows increased range of 32-bit constants to be loaded directly into registers
ARM Architecture – Program Status Word

- **Condition code flags**
  - **N** = Negative result from ALU
  - **Z** = Zero result from ALU
  - **C** = ALU operation Carried out
  - **V** = ALU operation overflowed

- **Sticky Overflow flag - Q flag**
  - Architecture 5TE/J only
  - Indicates if saturation has occurred

- **J bit**
  - Architecture 5TEJ only
  - **J = 1**: Processor in Jazelle state

- **Interrupt Disable bits.**
  - **I** = 1: Disables the IRQ.
  - **F** = 1: Disables the FIQ.

- **T Bit**
  - Architecture xT only
  - **T = 0**: Processor in ARM state
  - **T = 1**: Processor in Thumb state

- **Mode bits**
  - Specify the processor mode
ARM Architecture – CPU Execution Modes

- User: unprivileged mode under which most tasks run
- FIQ: entered when a high priority (fast) interrupt is raised
- IRQ: entered when a low priority (normal) interrupt is raised
- Supervisor: entered on reset and when a Software Interrupt instruction is executed
- Abort: used to handle memory access violations
- Undef: used to handle undefined instructions
- System: privileged mode using the same registers as user mode
Conclusion – Almost

- There is no magic solution for all discussed problems for all use cases
- It is necessary to combine discussed techniques and optimize the mix according to intended CPU area of use (the highest computational power/power efficient)
- Use of heterogeneous systems for high performance computation – vector units, GPU, FPGA accelerators
Why Instruction Set Architecture Matters

• Why can’t Intel sell mobile chips?
  99%+ of mobile phones/tablets are based on ARM’s v7/v8 ISA

• Why can’t ARM partners sell servers?
  99%+ of laptops/desktops/servers are based on the AMD64 ISA (over 95%+ built by Intel)

• How can IBM still sell mainframes?
  IBM 360 is the oldest surviving ISA (50+ years)

ISA is the most important interface in a computer system
ISA is where software meets hardware. (SiFive/RISC-V)
ARM 64-bit – AArch64

- Calling uses LR, no register banking, ELR for exceptions
- PC is separate register (not included in general purpose registers file)
- 31 64-bit registers R0 to R30 (R30 = X30 ≈ LR)
  - Symbol Wn (W0) used for 32-bit access, Xn (X0) for 64-bit
  - Reg. code 31 same zero role as MIPS 0, WZR/XZR in code
  - Reg. code 31 special meaning as WSP, SP for some opcodes
- Immediate operand 12-bit with optional LS 12 for arithmetics operations and repetitive bit masks generator for logic ones
- 32-bit operations ignores bits 32–63 for source and zeros these in the destination register
AArch64 – Branches and Conditional Operations

- Omitted conditional execution in all instructions as well as Thumb IT mechanism
- Conditional register retain, CBNZ, CBZ, TBNZ, TBZ added
- Only couple of conditional instructions
  - add and sub with carry, select (move C?A:B)
  - set 0 and 1 (or -1) according to the condition evaluation
  - conditional compare instruction
- 32-bit and 64-bit multiply and divide (3 registers), multiply with addition $64 \times 64 + 64 \rightarrow 64$ (four registers), high bits 64 to 127 from $64 \times 64$ multiplication
AArch64 – Memory Access

- 48+1 bit address, sign extended to 64 bits
- Immediate offset can be multiplied by access size optionally
- If register is used in index role, it can be multiplied by access size and can be limited to 32 bits
- PC relative ±4GB can be encoded in 2 instructions
- Only pair of two independent registers LDP and STP (omitted LDM, STM), added LDNP, STNP
- Unaligned access support
- LDX/STX(RBHP) for 1,2,4,8 and 16 bytes exclusive access
AArch64 – Address Modes

- **Simple register (exclusive)**
  
  \[\text{base}\{,#0\}\]

- **Offset**
  
  \[\text{base}\{,#imm\}\] – Immediate Offset

  \[\text{base},Xm\{,LSL \#imm\}\] – Register Offset

  \[\text{base},Wm,(S|U)XTW \{\#imm\}\] – Extended Register Offset

- **Pre-indexed**
  
  \[\text{base},\#imm]\!

- **Post-indexed**
  
  \[\text{base},\#imm\]

- **PC-relative (literal) load**
  
  label

<table>
<thead>
<tr>
<th></th>
<th>Sign</th>
<th>Scaling</th>
<th>WBctr</th>
<th>LD/ST type</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>LDX, STX, acquire, release</td>
</tr>
<tr>
<td>9</td>
<td>signed</td>
<td>scaled</td>
<td>option</td>
<td>reg. pair</td>
</tr>
<tr>
<td>10</td>
<td>signed</td>
<td>unscaled</td>
<td>option</td>
<td>single reg.</td>
</tr>
<tr>
<td>12</td>
<td>unsig.</td>
<td>scaled</td>
<td>no</td>
<td>single reg.</td>
</tr>
</tbody>
</table>
Apple A12Z Bionic – 64-bit ARM-based

- People who are really serious about software should make their own hardware. *Alan Kay*
- **Apple A12Z**, 8 cores (ARM big.LITTLE: 4 "big" Vortex + 4 "little" Tempest), Max. 2.49 GHz, ARMv8.3-A
- **Cache L1 128 KB instruction, 128 KB datam L2 8 MB**
- **GPU Apple designed 8-Core**
Fujitsu – Supercomputer Fugaku – A64FX, 2020 TOP500 #1

- Combine Armv8.2-A (AArch64 only) with Fujitsu supercomputer technology, SPARC64 V till now
- 48 computing cores + 4 assistant cores, SVE 512-bit wide SIMD
- HBM2 32GiB, 7nm FinFET, 8,786M transistors
- Tofu 6D Mesh/Torus, 28Gbps x 2 lanes x 10 ports, PCIe

Source: Fujitsu High Performance CPU for the Post-K Computer, 2018
RISC-V – Optimize and Simplify RISC Again

- Patterson, Berkeley RISC 1984 → initiation of RISC era, evolved into SPARC (Hennessy MIPS, Stanford University)
- Commercialization and extensions results in too complex CPUs again, with license and patents preventing even original investors to use real/actual implementations in silicon to be used for education and research
- MIPS is model architecture for prevalent amount of base courses and implementation of similar processor is part of follow up courses (A4M36PAP)
- Krste Asanovic and other Dr. Patterson's students initiated development of new architecture (start of 2010)
- BSD Licence to ensure openness in future
- Supported by GCC, binutils., Linux, QEMU, etc.
- Simpler than SPAC, more like MIPS but optimized on gate level load (fanout) and critical paths lengths in future designs
- Some open implementations already exists Rocket (SiFive, BOOM), project lowRISC contributes to research in security area, in ČR Codasip
- Already more than 20 implementations in silicon
RISC-V – Architecture Specification

- ISA specification can be found at http://riscv.org/
  - Andrew Waterman, Yunsup Lee, David Patterson, Krste Asanovic
  - Not only architecture description but even choices analysis with pro&cons of each selection and cost source description/analysis of alternatives

- classic design, 32 integer registers, the first tied to zero, regsrc1, regsrc2, regdest operands, uniqueness, rule kept strictly even for SaveWord, leads to non-continuous immediate operands encoding, PC not part of base register file, PC-relative addressing

- variants for 32, 64 a 128-bit registers and address-space defined

- high code density (16-bit instruction encoding variant planned)

- encoding reserves space for floating point (single, double, quad) and multimedia SIMD instructions systematic way, etc.
## RISC-V – Registers

### Integer registers

<table>
<thead>
<tr>
<th>XLEN-1</th>
<th>0</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x0 / zero</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>x31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>XLEN</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variant</th>
<th>XLEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>RV32</td>
<td>32</td>
</tr>
<tr>
<td>RV64</td>
<td>64</td>
</tr>
<tr>
<td>RV128</td>
<td>128</td>
</tr>
</tbody>
</table>

### Floating point registers

<table>
<thead>
<tr>
<th>FLEN-1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>f0</td>
<td></td>
</tr>
<tr>
<td>f1</td>
<td></td>
</tr>
<tr>
<td>f2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>f29</td>
<td></td>
</tr>
<tr>
<td>f30</td>
<td></td>
</tr>
<tr>
<td>f31</td>
<td></td>
</tr>
<tr>
<td>FLEN</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variant</th>
<th>FLEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>32</td>
</tr>
<tr>
<td>D</td>
<td>64</td>
</tr>
<tr>
<td>Q</td>
<td>128</td>
</tr>
</tbody>
</table>

### Floating-point control and status register

<table>
<thead>
<tr>
<th>31</th>
<th>8 7</th>
<th>5 4 3 2 1 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reserved</td>
<td>Rounding Mode (frm)</td>
<td>Accrued Exceptions (fflags)</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>1 1 1 1 1 1</td>
</tr>
</tbody>
</table>

Source: [https://riscv.org/specifications/](https://riscv.org/specifications/)
RISC-V – Instruction Length Encoding

<table>
<thead>
<tr>
<th>Instruction Length</th>
<th>Encoding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>16-bit (aa ≠ 11)</td>
<td>xxxxxxxxxxaxaa</td>
<td></td>
</tr>
<tr>
<td>32-bit (bbb ≠ 111)</td>
<td>xxxxxxxxxxbbb11</td>
<td></td>
</tr>
<tr>
<td>48-bit</td>
<td>xxxxxxxxxx011111</td>
<td></td>
</tr>
<tr>
<td>64-bit</td>
<td>xxxxxxxxxx011111</td>
<td></td>
</tr>
<tr>
<td>(80+16*nnn)-bit, nnn ≠ 11</td>
<td>xnnnxxxxx111111</td>
<td>Reserved for ≥192-bits</td>
</tr>
<tr>
<td>Reserved for ≥192-bits</td>
<td>x111xxxxx111111</td>
<td>Reserved for ≥192-bits</td>
</tr>
</tbody>
</table>

Address:
- base+4
- base+2
- base

Source: https://riscv.org/specifications/
RISC-V – 32-bit Instructions Encoding

Source: [https://riscv.org/specifications/](https://riscv.org/specifications/)
BOOM Superscalar RISC-V into Rocket Chip

Fetch → Decode & Rename → Rename Map Tables & Freelist → Issue Window → ROB → Commit

Unified Physical Register File (PRF) → ALU → FPU

in-order front-half

out-of-order back-half

Main developer: Christopher Celio

9k source lines + 11k from Rocket

RISC-V – HiFive Unleashed

- SiFive FU540-C000 (built in 28nm)
  - 4+1 Multi-Core Coherent Configuration, up to 1.5 GHz
  - 4x U54 RV64GC Application Cores with Sv39 Virtual
- Memory Support
- 1x E51 RV64IMAC Management Core
- Coherent 2MB L2 Cache
- 64-bit DDR4 with ECC
- 1x Gigabit Ethernet
- 8 GB 64-bit DDR4 with ECC
- Gigabit Ethernet Port
- 32 MB Quad SPI Flash
- MicroSD card for removable storage
- FMC connector for future expansion with add-in card
RISC-V – HiFive Unleashed

[Diagram showing the architecture of a HiFive board with various components such as cores, caches, memory controllers, and peripheral interfaces.]
More RISC-V projects

- Libre RISC-V [https://libre-riscv.org/](https://libre-riscv.org/)
  - Quad-core 28nm RISC-V 64-bit (RISCV64GC core with Vector SIMD Media / 3D extensions)
  - 300-pin 15x15mm BGA 0.8mm pitch
  - 32-bit DDR3/DDR3L/LPDDR3 memory interface
- More RISC-V resources
  - [https://riscv.org/](https://riscv.org/)
  - RISC-V YouTube channel [https://www.youtube.com/channel/UC5gLmcFuvdGbajs4VL-WU3g](https://www.youtube.com/channel/UC5gLmcFuvdGbajs4VL-WU3g)