

東北大学 工学部 機械知能・航空工学科
2017年度 5セメスター・クラスC3 D1 D2 D3

計算機工学

5. 命令セットアーキテクチャ (教科書6.1節, 6.2節)

大学院情報科学研究科

鏡 慎吾

<http://www.ic.is.tohoku.ac.jp/~swk/lecture/>

計算機の基本構成

メモリ

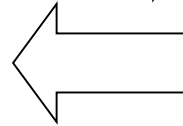
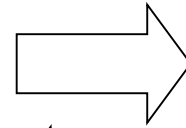
データ領域

データ
データ
データ
...

プログラム領域

命令
命令
命令
...

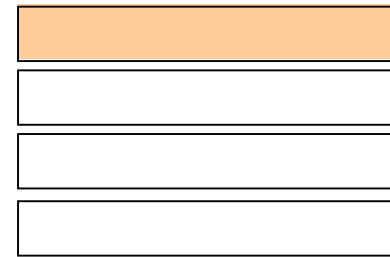
load



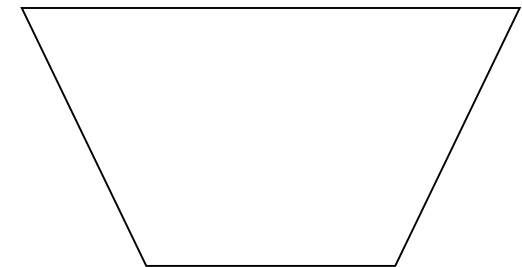
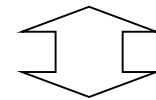
store

プロセッサ

レジスタ



PC



演算器 (ALU)

計算機の基本動作

- プロセッサは、メモリのプログラム領域から命令をアドレス順に読み出して実行する
- 演算は ALU (Arithmetic Logic Unit) が行う
- 必要に応じて、メモリとプロセッサ内のレジスタとの間でデータを移動する
 - 演算を行うには、少なくとも瞬時にはプロセッサ内でデータを記憶しておく必要がある
 - 最近のほとんどのプロセッサは、メモリ内のデータではなくレジスタ内のデータを演算の対象とする (∵ メモリはプロセッサに対して遅いため)
 - load: メモリ → レジスタ
 - store: メモリ ← レジスタ
- PC (Program Counter) と呼ばれる特殊レジスタに、次に実行する命令のアドレスが保存されている
 - PCの内容は命令実行ごとに更新される

命令セットアーキテクチャ

- プロセッサが実行できる命令の集合を命令セット (instruction set) と呼ぶ. 実際には, プログラムから使用できるレジスタの種類, メモリアドレスの指定方法なども含めて命令セットと呼ぶのが通常である
- ソフトウェアから見たときに, そのプロセッサがどんなものであるかは, 命令セットによって決まる. この観点から見たアーキテクチャを命令セットアーキテクチャ (Instruction Set Architecture: ISA) と呼ぶ
- それに対し, ある命令セットアーキテクチャをどのような回路でどのような動作タイミングで実現するかという観点から見たアーキテクチャをマイクロアーキテクチャと呼ぶ
 - 同じ ISA に対して多数のマイクロアーキテクチャがあり得る

命令セットアーキテクチャの例

- x86 (IA-32, i386)
いわゆる PC 用のCPUで採用. PC以外にも広く利用される.
- PowerPC
以前の Macintosh. PlayStation 3, Xbox 360, Nintendo Wii
- SPARC
Sun Microsystems のワークステーション, 各種組み込み機器
- MIPS
Silicon Graphics, Sony, NEC のワークステーション,
初代 PlayStation, Nintendo 64, PSP, 各種組み込み機器, 携帯機器など
- ARM
携帯機器・携帯電話の多く, ゲームボーイアドバンス, Nintendo DS, DSi,
Nintendo Switch
- SuperH (SH)
各種組み込み機器, 携帯機器, セガサターン, ドリームキャスト

注: 厳密な命令セットアーキテクチャ名としては, さらに細かく分類される
(例えば MIPS I, MIPS II, MIPS32, MIPS64...)

歴史的な経緯

- 当初は、計算機的设计と具体的な製品は 1 対 1 対応
- IBM System/360 (1964) で、統一的なアーキテクチャによる「計算機ファミリ」の概念が現れる
- 初の商用マイクロプロセッサ Intel 4004 (1971) 以降、計算機本体とは独立の「部品」としてプロセッサを扱えるようになる(計算機メーカーとプロセッサメーカーの分離)
- 1980年代頃、RISCへの転回
 - **RISC** (Reduced Instruction Set Computer):
命令セットを簡素化し、回路を単純化することで高速化
 - **CISC** (Complex Instruction Set Computer):
RISC に対して従来のアーキテクチャをこう呼んだ

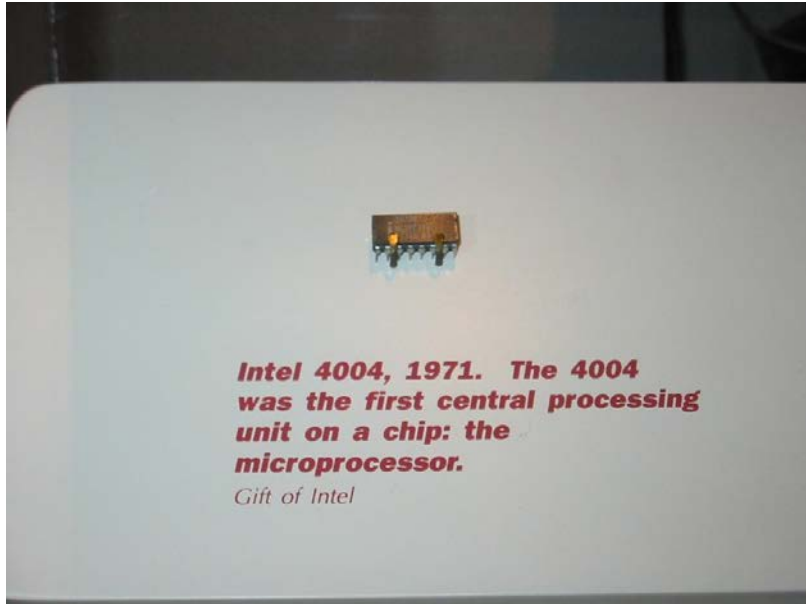
IBM System/360



http://upload.wikimedia.org/wikipedia/commons/8/8d/Bundesarchiv_B_145_Bild-F038812-0014%2C_Wolfsburg%2C_VW_Autowerk.jpg

「コンピュータアーキテクチャ」という概念をおそらく最初に明確に導入した商用計算機. オペレーティングシステム(OS)を最初に導入した商用計算機でもある

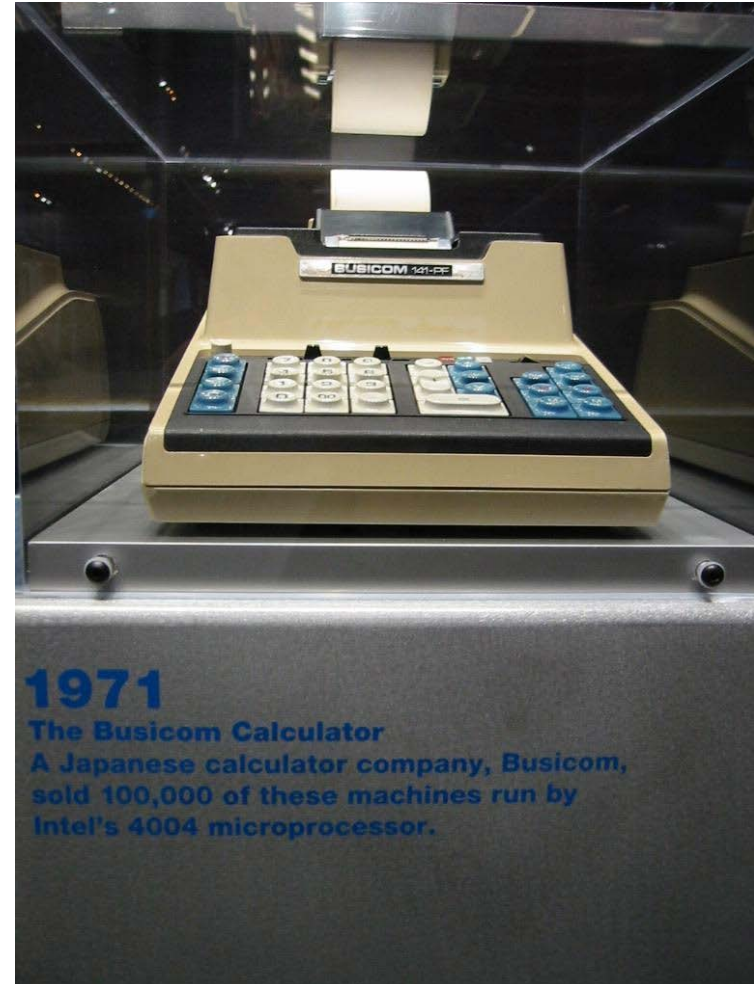
intel 4004



Intel 4004, 1971. The 4004 was the first central processing unit on a chip: the microprocessor.

Gift of Intel

(American History Museum)



1971

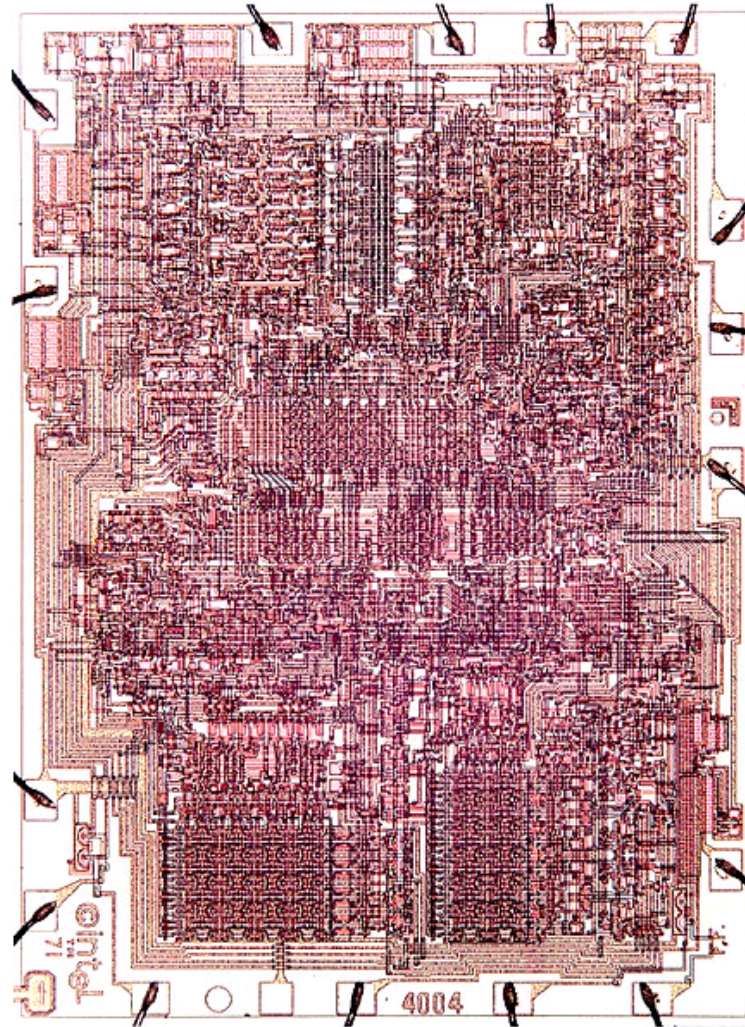
The Busicom Calculator

A Japanese calculator company, Busicom, sold 100,000 of these machines run by Intel's 4004 microprocessor.

Busicom 141-PF

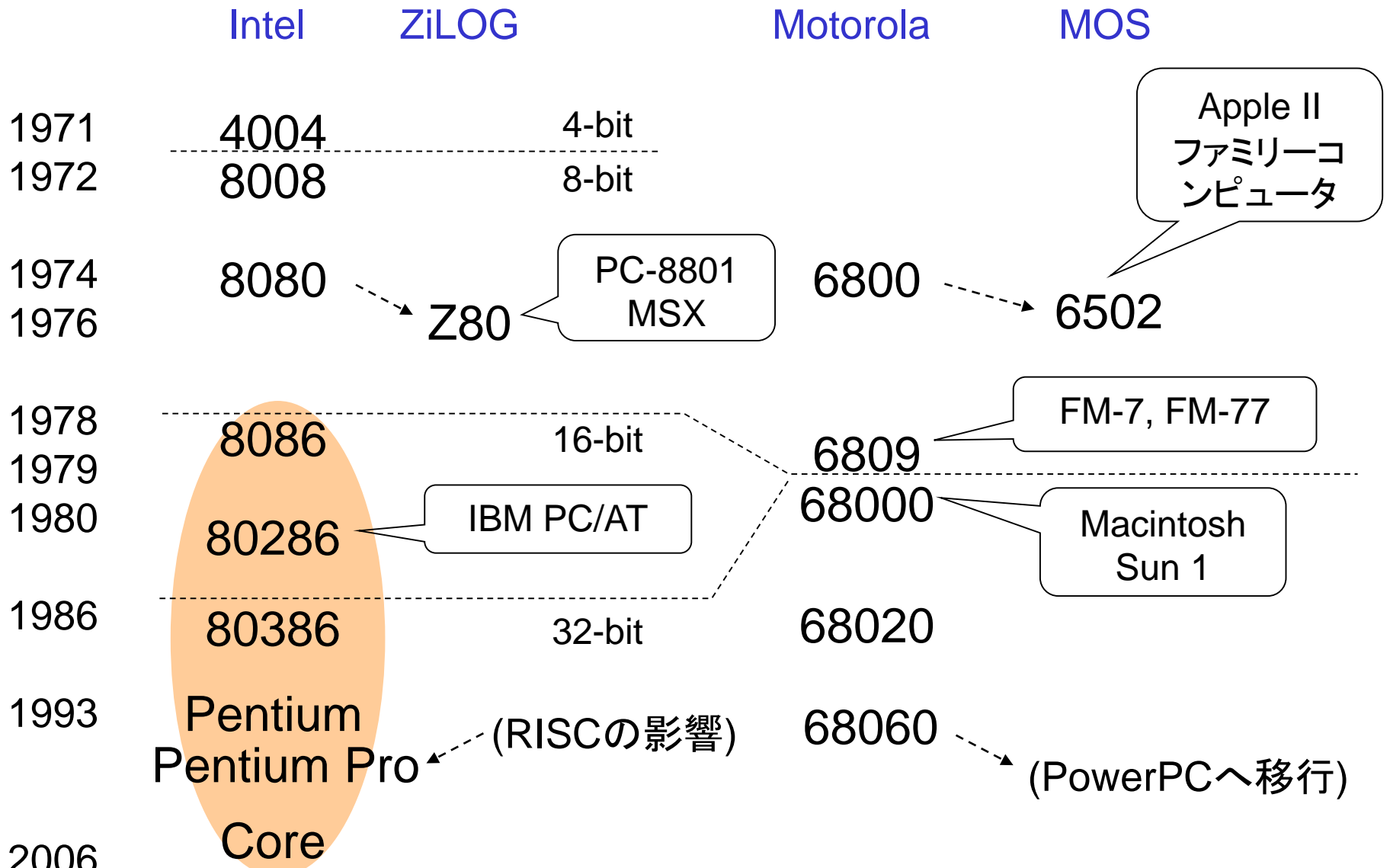
(Intel Museum, Santa Clara)

intel 4004



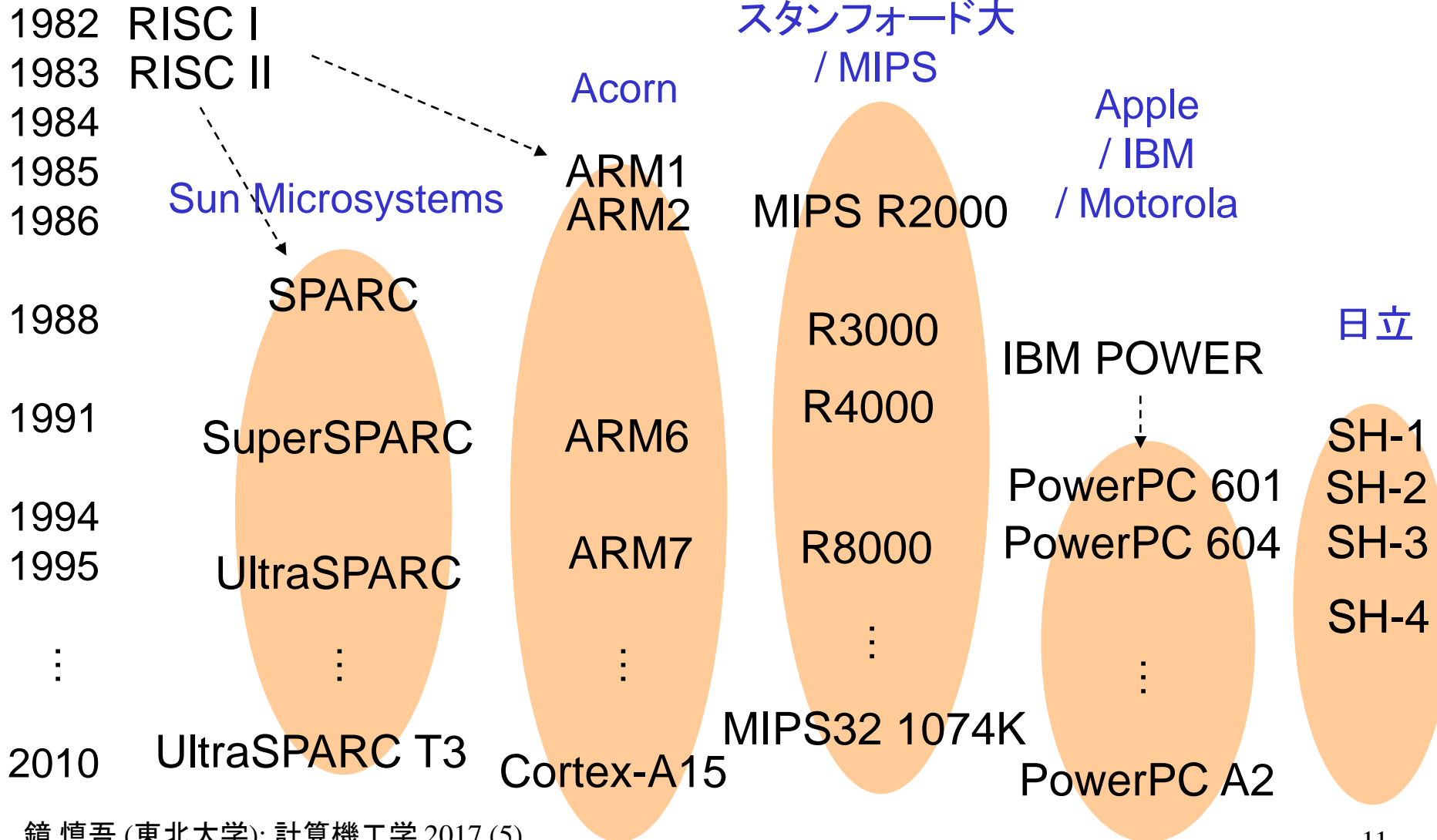
http://news.com.com/1971+Intel+4004+processor/2009-1006_3-6038974-3.html

マイクロプロセッサの系譜 (CISC)



マイクロプロセッサの系譜 (RISC)

カリフォルニア大バークレイ校



ゲーム機用プロセッサ

任天堂 ファミリーコンピュータ (1983), NEC PCエンジン (1987): 6502

セガ マークIII (1985): Z80

セガ メガドライブ (1988): 68000 + Z80

任天堂 スーパーファミコン (1990): 65C816 (6502の後継)

セガサターン (1994): SH-2

ソニー PlayStation (1994): MIPS R3000

任天堂 NINTENDO64 (1996): MIPS R4300

セガ ドリームキャスト (1998): SH-4

ソニー PlayStation2 (2000): EmotionEngine (MIPS R5900ベース)

任天堂 ゲームキューブ (2001): PowerPC 750

マイクロソフト Xbox (2001): Mobile Celeron (Pentium IIIベース)

マイクロソフト Xbox 360 (2005): Xenon (PowerPCベース)

ソニー PlayStation3 (2006): Cell (PowerPCベース)

任天堂 Wii (2006): Broadway (PowerPCベース)

任天堂 Wii U (2010): Espresso (Powerベース)

ソニー PlayStation4 (2013): AMD Jaguar (x86ベース)

マイクロソフト Xbox One (2013): AMD Jaguar (x86ベース)

任天堂 Switch (2017): NVIDIA Tegra X1 (ARM Cortex-A57/A53ベース)

携帯電話・タブレット端末用プロセッサ

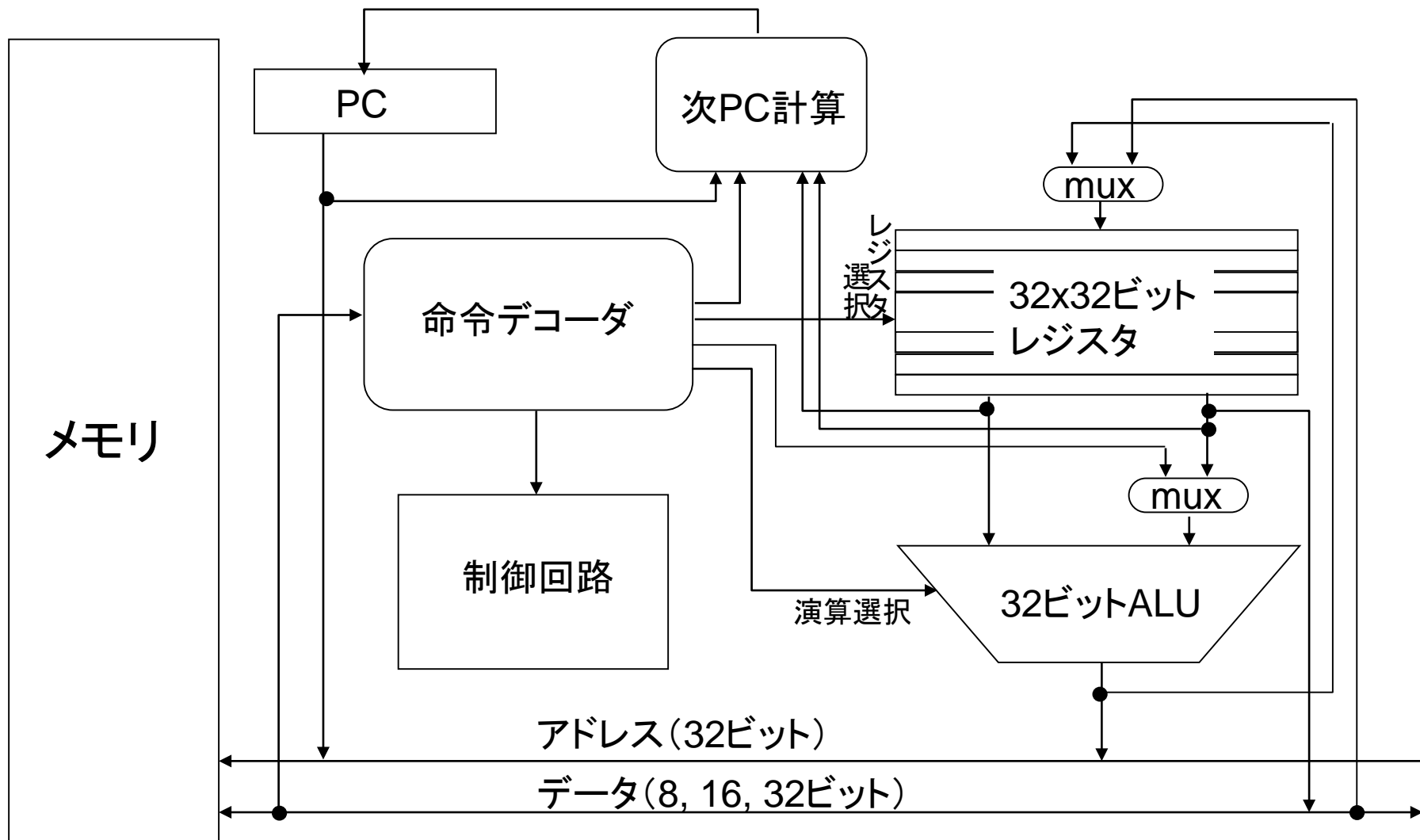
- Qualcomm Snapdragon (ARM)
- MediaTek MT6xxx (ARM)
- HiSilicon Kirin (ARM)
- Samsung Exynos (ARM)
- Apple A7, A8, ... (ARM)

- Texas Instruments OMAP (ARM)
- NVIDIA Tegra (ARM)
- Intel Atom (x86)
- ルネサス SH-Mobile (SuperH)

MIPSアーキテクチャ

- この講義では, MIPS I アーキテクチャを取り上げて計算機の動作を学ぶ
 - 現代的なアーキテクチャの基本形ともいえる構成
 - 組み込み機器を中心に, 世界中で使われている
 - 世界中の大学の講義で取り上げられている
- 特徴
 - 32本 × 32ビット汎用レジスタ
 - 32ビットALU
 - 32ビットのメモリアドレス空間
 - PCは汎用レジスタとは別に存在 (勝手にロード・ストアできない)

MIPSの構造



(参考) MIPSシミュレータ SPIM

参考書 (パターンソン・ヘネシー) でも紹介されているシミュレータ SPIM を使うと, MIPSの動作を確認することができる.

<http://spimsimulator.sourceforge.net/>

- UNIX, MacOS, Windows で動作

最低限の動かし方:

- File → Reinitialize and Load File でアセンブリ言語ファイルを開く
- Simulator → Run/Continue (F5) で実行
- あるいは Simulator → Single Step (F10) で1行ずつ実行

講義に対応したサンプルプログラム:

- <http://www.ic.is.tohoku.ac.jp/~swk/lecture/>


```

Int Regs [16]
PC      = 400048
EPC     = 0
Cause   = 0
BadVAddr = 0
Status  = 3000ff10

HI      = 0
LO      = 0

R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 4
R3 [v1] = 0
R4 [a0] = 1
R5 [a1] = 7ffff090
R6 [a2] = 7ffff098
R7 [a3] = 0
R8 [t0] = 2
R9 [t1] = 3
R10 [t2] = 5
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0

```

レジスタの表示

```

R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7fff8000
R30 [s8] = 0
R31 [ra] = 400018

```

QtSpim
File Simulator Registers Text Segment Data Segment Window

```

Text
User Text Segment [00400000]..[00440000]
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[00400014] 0c100009 jal 0x00400024 [main] ; 188: jal main
[00400018] 00000000 nop ; 189: nop
[0040001c] 3402000a ori $2, $0, 10 ; 191: li $v0 10
[00400020] 0000000c syscall ; 192: sy
[00400024] 27bd8f74 addiu $29, $29, -28812 ; 4: addi
[00400028] 34080002 ori $8, $0, 2 ; 5: ori
[0040002c] afa80000 sw $8, 0($29) ; 6: sw $
[00400030] 34080003 ori $8, $0, 3 ; 7: ori
[00400034] afa80004 sw $8, 4($29) ; 8: sw $
[00400038] 8fa80000 lw $8, 0($29) ; 11: lw
[0040003c] 8fa90004 lw $9, 4($29) ; 12: lw
[00400040] 01095021 addu $10, $8, $9 ; 13: addu $t2, $t0, $t1
[00400044] afaa0008 sw $10, 8($29) ; 14: sw $t2, 8($sp)
[00400048] 27bd708c addiu $29, $29, 28812 ; 16: addiu $sp, $sp, 0x708c
[0040004c] 03e00008 jr $31 ; 17: jr $ra

```

プログラムの表示
「syscall」まではシステムが用意した初期化コード

```

Data
User data segment [10000000]..[10040000]
[10000000]..[1003ffff] 00000000
[80000188]
[8000018c]
registers
User Stack [7fff8000]..[80000000]
[7fff8000] 00000002 00000003 00000005 00000000 . . . . .
[7fff8010]..[7ffff08b] 00000000
[7ffff08c] 00000001 . . . . .
[7ffff090] 7ffff17b 00000000 7fffffe1 7fffffbc { . . . . .
[7ffff0a0] 7fffff85 7fffff49 7fffff18 7fffff06 . . . . . I . . . . .
[7ffff0b0] 7ffffee2 7ffffebb 7ffffe78 7ffffe64 . . . . . x . . . . . d . . . . .
[7ffff0c0] 7ffffe57 7ffffe49 7ffffe31 7ffffe24 W . . . . I . . . . 1 . . . . $ . . . .
[7ffff0d0] 7ffffe10 7ffffde8 7ffffdd5 7ffffd8b . . . . .
[7ffff0e0] 7ffffd41 7ffffd2a 7ffffd1c 7ffff678 A . . . . * . . . . x . . . .
[7ffff0f0] 7ffff63a 7ffff608 7ffff5ed 7ffff5d0 ; . . . . .

```

メモリ値の表示

SPIMに読み込ませるアセンブリ言語ファイルの例

```
.text
.globl main
main:
    addu $sp, $sp, -0x300
    or $t0, $zero, 1
    sw $t0, 0($sp)
```

```
###
    addu $t0, $sp, 4
    lw $t1, 0($sp)
    sll $t1, $t1, 2
    addu $t0, $t0, $t1
    or $t2, $zero, 300
    sw $t2, 0($t0)
```

```
###
    addu $sp, $sp, 0x300
    jr $ra
```

おまじない. 自分のプログラムは
main ラベルから始める.

レジスタやメモリ等の初期化.
わからなくても気にしない.

講義中の説明で理解して欲しい部分.

main の終了.