Architecture matérielle

Xavier Urbain

2018/2019

Il y aura une page web...

Surveillez vos emails (rappels, informations. . .)

Vous m'écrivez ? ⇝ "ASR4" + groupe dans le sujet politesse de base
depuis `@etu.univ-lyon1.fr`

Modalités de contrôle des connaissances : essentiellement CC
Plus ou moins une interro "surprise" par TD

Sources et références :

- Hennessy-Patterson : *Computer Architecture: A Quantitative Approach*
- Nisan & Schocken : *The Elements of Computer Systems*
  `www.nand2tetris.org`

Également cours de D. Étiemble et cours de N. Louvet

Ordinateurs ?

Programme : *grosso modo* expression finie (et si possible succincte) d'une façon d'obtenir un résultat effectivement

Ordinateur : dispositif physique pour

- Réaliser les calculs. . .
- Sur des données. . .

décrits par le programme

Ordinateurs. . .

Partout

- Supercalculateurs
- PC 2000 ⇝ 150 000 000
- Véhicules, embarqué 2000 ⇝ 6 000 000 000
- vidéo, signal, etc. 2000 ⇝ 600 000 000

Ordinateurs. . . Une vieille histoire. . .

- Pascaline mi-XVII^{e}, machine de Leibniz début XVIII^{e}
- Machine analytique de Babbage. . . (XIX^{e})
- ENIAC (1946) tubes à vide
  - 30t, 72m^2, 140kW
  - environ 350 multiplications par s. (10 chiffres)
- CDC, Honeywell (1960-1968) transistors
- DEC (1969-1977) circuits intégrés
- IBM-PC, Apple II 1978-2000 ? LSI, VLSI
- Portable actuel parallélisme. . .
  - 1,5kg, 90W, 2 cœurs, milliards d'opérations par s. (64 chiffres). . .

Ordinateurs. . . Machine de von Neumann

John von Neumann (1903-1957)

Ordinateur : calculs / données (/ programmes)

Machine de von Neumann :

- Centre pour calculs unité centrale (UC)
- Centre pour données **et programmes** mémoire centrale

qui communiquent. . . (bus)

Programmes en mémoire, briques de bases : instructions

Pas d'évolution depuis 20 ans. . .

Ordinateurs. . . Mémoires

Mémoires à accès aléatoire RAM : *temps d'accès identique partout*

Chères mais rapides SRAM : registres

- Peu
- Souvent nommés
- Certains spécialisés (0, résultat comparaison, CP, RI. . .)

. . . Hiérarchie de mémoires cache. . .

Peu chères mais lentes DRAM : mémoire principale

- Beaucoup
- Par adresse

Niveau programmeur : choses compliquées. . . évoluées. . .

Niveau physique : très peu de choses !
⇝ Représenter l'information et
⇝ Traiter l'information représentée

Information :
Distinction d'un état donné parmi plusieurs, à un instant donné

Physique : on sait faire avec 2 états Chargé ou pas (1V)

2 états ⇝ codage binaire
1 bloc (bit) : 2 états 2 blocs : 4 états n blocs ?

⋆ Valeur d'un bit : connaissance d'un état parmi deux

Bloc de 8 bits : octet Byte

Du coup, quand on parle de kilo octets ?

Préfixe	symbole	puissance de 10
péta	P	10^{15}
téra	T	10^{12}
giga	G	10^{9}
méga	M	10^{6}
kilo	k	10^{3}
–	–	10^{0}
milli	m	10^{-3}
micro	μ	10^{-6}
nano	n	10^{-9}

Du coup, quand on parle de kilo octets ? Ici binaire !

Ex. 1024 octets souvent abusivement noté 1ko.

Norme CEI-60027-2 de *préfixes binaires*

Préfixe courant	Symbole courant	Préfixe standardisé	Symbole standardisé	Puissance de 2	Puissance de 10 la plus proche
téra	T	tébi	Ti	2^{40}	10^{12}
giga	G	gibi	Gi	2^{30}	10^{9}
méga	M	mébi	Mi	2^{20}	10^{6}
kilo	k	kibi	ki	2^{10}	10^{3}
–	–	–	–	2^{0}	10^{0}

Ex. 1024 octets = 1 kibi-octet = 1kio.

Instructions en mémoire stockées sous forme d'un ou plusieurs mots en binaire

- code-opération ou opcode désignant l'opération
- aucune, une ou plusieurs opérandes : valeurs ou emplacement des sources, emplacement du résultat.

Ex. sur une certaine machine, sur 16 bits :

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
opcode				adresse											

Exemples d'opcodes hypothétiques:

- 0001 : charger le mot dont l'adresse est donnée dans ACC ;
- 0010 : stocker le mot contenu dans ACC à l'adresse donnée.

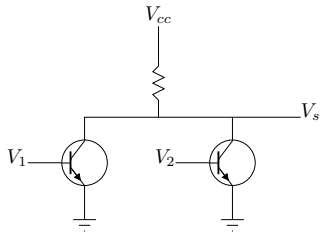
Zoom. . . Niveau 0

Portes logiques — taille qq molécules d'eau

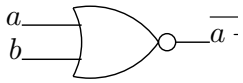
Traitements de données élémentaires.

Il y a 50 ans. . . avec tubes à vide
Aujourd'hui avec transistors (nMOS, pMOS, CMOS)

Entrées : 0 ou 1, $\top$ ou $\bot$ enfin 2 valeurs distinctes. . .
Sortie : application fonction logique simple NON, ET, OU. . .



V_1	V_2	V_s
0	0	1
0	1	0
1	0	0
1	1	0



Zoom. . . Niveau 1

Organisation des portes → circuits spécialisés

Micro-architecture : *matériel pour exécution du langage machine*

- Circuits pour opérations de base (LM), logique et calcul : UAL
- Chef d'orchestre : circuits de contrôle métronome : horloge

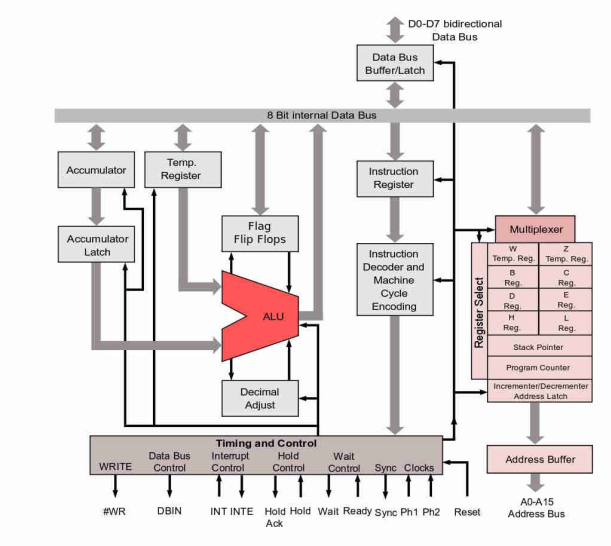
composent le processeur

Cycle d'éxécution d'une instruction :

Lec I	Lec O	Exé	Ran

Ici 4 cycles d'horloge, *dépend du processeur*

Zoom. . . Niveau 1



Zoom. . . Niveau 2

Informations sur le matériel si on veut programmer à ce niveau

Architecture logicielle : *ensemble des informations visibles au prog en LM*

Typiquement :

- Ensemble des instructions (jeu d'instructions) + format
- Organisation de la mémoire (adressage)
- Représentations élémentaires (entiers, flottants. . .) on est en binaire !

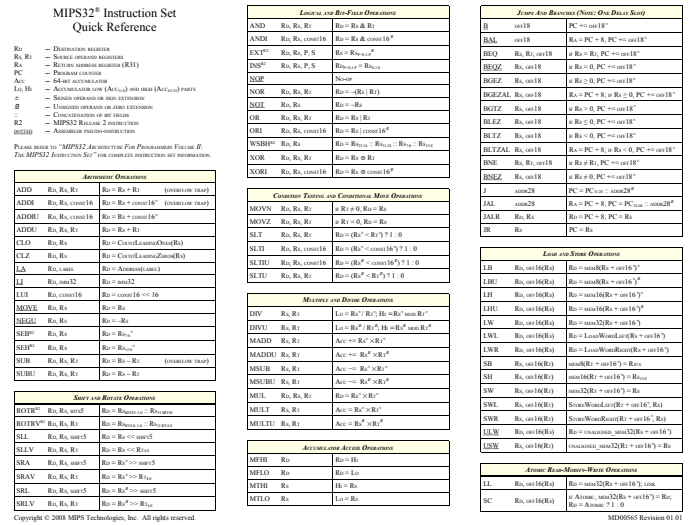
Même architecture logicielle pour différentes micro-archi :

Pentium I TRÈS différent de Pentium IV mais même archi x86

Zoom. . . Niveau 2

instruction	action	nzp	codage en langage machine
			opcode (F E D C) / arguments (B A 9 8 7 6 5 4 3 2 1 0)
NOT DR,SR	DR <- not SR	*	1001 / DR / SR / 111111
ADD DR,SR1,SR2	DR <- SR1 + SR2	*	0001 / DR / SR1 / 0 / 00 / SR2
ADD DR,SR1,Imm5	DR <- SR1 + SEXT(Imm5)	*	0001 / DR / SR1 / 1 / Imm5
AND DR,SR1,SR2	DR <- SR1 and SR2	*	0101 / DR / SR1 / 0 / 00 / SR2
AND DR,SR1,Imm5	DR <- SR1 and SEXT(Imm5)	*	0101 / DR / SR1 / 1 / Imm5
LEA DR,label	DR <- PC + SEXT(PCoffset9)	*	1110 / DR / PCoffset9
LD DR,label	DR <- mem[PC + SEXT(PCoffset9)]	*	0010 / DR / PCoffset9
ST SR,label	mem[PC + SEXT(PCoffset9)] <- SR		0011 / SR / PCoffset9
LDR DR,BaseR,Offset6	DR <- mem[BaseR + SEXT(Offset6)]	*	0110 / DR / BaseR / Offset6
STR SR,BaseR,Offset6	mem[BaseR + SEXT(Offset6)] <- SR		0111 / SR / BaseR / Offset6
LDI DR,label	DR <- mem[mem[PC + SEXT(PCoffset9)]]	*	1010 / DR / PCoffset9
STI SR,label	mem[mem[PC + SEXT(PCoffset9)]] <- SR		1011 / SR / PCoffset9
BR[n][z][p] label	Si (cond) PC <- PC + SEXT(PCoffset9)		0000 / n z p / PCoffset9
NOP	No Operation		0000 / 0 0 0 / 000000000
JMP BaseR	PC <- BaseR		1100 / 000 / BaseR / 000000
RET (JMP R7)	PC <- R7		1100 / 000 / 111 / 000000
JSR label	R7 <- PC; PC <- PC + SEXT(PCoffset11)		0100 / 1 / PCoffset11
JSRR BaseR	R7 <- PC; PC <- BaseR		0100 / 0 / 00 / BaseR / 000000
RTI	cf. interruptions		1000 / 000000000000
TRAP Trapvect8	R7 <- PC; PC <- mem[Trapvect8]		1111 / 0000 / Trapvect8
Réservé			1101

Zoom. . . Niveau 2



MIPS32® Instruction Set Quick Reference

Rd — Destination register
Rs, Rt — Source operand registers
Ra — Return address register (R31)
PC — Program counter
Acc — 64-bit accumulator
Lo, Hi — Accumulator low ($Acc_{31:0}$) and high ($Acc_{63:32}$) parts
+ — Signed operand or sign extension
0 — Unsigned operand or zero extension
|| — Concatenation of bit fields
R2 — MIPS32 Release 2 instruction
pseudo — Assembler pseudo-instruction

Please refer to *"MIPS32 Architecture For Programmers Volume II: The MIPS32 Instruction Set"* for complete instruction set information.

Arithmetic Operations

ADD	Rd, Rs, Rt	Rd = Rs + Rt	(overflow trap)
ADDI	Rd, Rs, const16	Rd = Rs + const16+	(overflow trap)
ADDIU	Rd, Rs, const16	Rd = Rs + const16+	
ADDU	Rd, Rs, Rt	Rd = Rs + Rt	
CLO	Rd, Rs	Rd = Count_Leading_Ones(Rs)	
CLZ	Rd, Rs	Rd = Count_Leading_Zeros(Rs)	
LA	Rd, label	Rd = Address(label)	
LI	Rd, imm32	Rd = imm32	
LUI	Rd, const16	Rd = const16 << 16	
MOVE	Rd, Rs	Rd = Rs	
NEGU	Rd, Rs	Rd = −Rs	
SEB[2]	Rd, Rs	Rd = Rs_{7:0}+	
SEH[2]	Rd, Rs	Rd = Rs_{15:0}+	
SUB	Rd, Rs, Rt	Rd = Rs − Rt	(overflow trap)
SUBU	Rd, Rs, Rt	Rd = Rs − Rt	

Shift and Rotate Operations

ROTR[2]	Rd, Rs, sa5	Rd = R_{sa5-1:0} :: R_{31:sa5}
ROTRV[2]	Rd, Rs, Rt	Rd = R_{Rt4:0} :: R_{31:Rt4}
SLL	Rd, Rs, sa5	Rd = Rs << sa5
SLLV	Rd, Rs, Rt	Rd = Rs << Rt4:0
SRA	Rd, Rs, sa5	Rd = Rs+ >> sa5
SRAV	Rd, Rs, Rt	Rd = Rs+ >> Rt4:0
SRL	Rd, Rs, sa5	Rd = Rs0 >> sa5
SRLV	Rd, Rs, Rt	Rd = Rs0 >> Rt4:0

Logical and Bit-Field Operations

AND	Rd, Rs, Rt	Rd = Rs & Rt
ANDI	Rd, Rs, const16	Rd = Rs & const16#
EXT[2]	Rd, Rs, P, S	Rd = Rs_{P+S-1:P}#
INS[2]	Rd, Rs, P, S	Rd_{P+S-1:P} = Rs_{S-1:0}
NOP		No-op
NOR	Rd, Rs, Rt	Rd = ~(Rs | Rt)
NOT	Rd, Rs	Rd = ~Rs
OR	Rd, Rs, Rt	Rd = Rs | Rt
ORI	Rd, Rs, const16	Rd = Rs | const16#
WSBH[2]	Rd, Rs	Rd = Rs_{23:16} :: Rs_{31:24} :: Rs_{7:0} :: Rs_{15:8}
XOR	Rd, Rs, Rt	Rd = Rs ⊕ Rt
XORI	Rd, Rs, const16	Rd = Rs ⊕ const16#

Conditional Testing and Conditional Move Operations

MOVN	Rd, Rs, Rt	if Rt ≠ 0, Rd = Rs
MOVZ	Rd, Rs, Rt	if Rt = 0, Rd = Rs
SLT	Rd, Rs, Rt	Rd = (Rs* < Rt*) ? 1 : 0
SLTI	Rd, Rs, const16	Rd = (Rs* < const16*) ? 1 : 0
SLTIU	Rd, Rs, const16	Rd = (Rs# < const16#) ? 1 : 0
SLTU	Rd, Rs, Rt	Rd = (Rs# < Rt#) ? 1 : 0

Multiply and Divide Operations

DIV	Rs, Rt	Lo = Rs* / Rt*; Hi = Rs* mod Rt*
DIVU	Rs, Rt	Lo = Rs# / Rt#; Hi = Rs# mod Rt#
MADD	Rs, Rt	Acc += Rs* ×Rt*
MADDU	Rs, Rt	Acc += Rs# ×Rt#
MSUB	Rs, Rt	Acc −= Rs* ×Rt*
MSUBU	Rs, Rt	Acc −= Rs# ×Rt#
MUL	Rd, Rs, Rt	Rd = Rs* ×Rt*
MULT	Rs, Rt	Acc = Rs* ×Rt*
MULTU	Rs, Rt	Acc = Rs# ×Rt#

Accumulator Access Operations

MFHI	Rd = Hi
MFLO	Rd = Lo
MTHI	Hi = Rs
MTLO	Lo = Rs

Jumps And Branches (None: One Delay Slot)

B	const18	PC += const18*
BAL	const18	Ra = PC + 8; PC += const18*
BEQ	Rs, Rt, const18	si Rs = Rt, PC += const18*
BEQZ	Rs, const18	si Rs = 0, PC += const18*
BGEZ	Rs, const18	si Rs ≥ 0, PC += const18*
BGEZAL	Rs, const18	Ra = PC + 8; si Rs ≥ 0, PC += const18*
BGTZ	Rs, const18	si Rs > 0, PC += const18*
BLEZ	Rs, const18	si Rs ≤ 0, PC += const18*
BLTZ	Rs, const18	si Rs < 0, PC += const18*
BLTZAL	Rs, const18	Ra = PC + 8; si Rs < 0, PC += const18*
BNE	Rs, Rt, const18	si Rs ≠ Rt, PC += const18*
BNEZ	Rs, const18	si Rs ≠ 0, PC += const18*
J	asm28	PC = PC_{31:28} :: asm28#
JAL	asm28	Ra = PC + 8; PC = PC_{31:28} :: asm28#
JALR	Rs, Rs	Rs = PC + 8; PC = Rs
JR	Rs	PC = Rs

Load and Store Operations

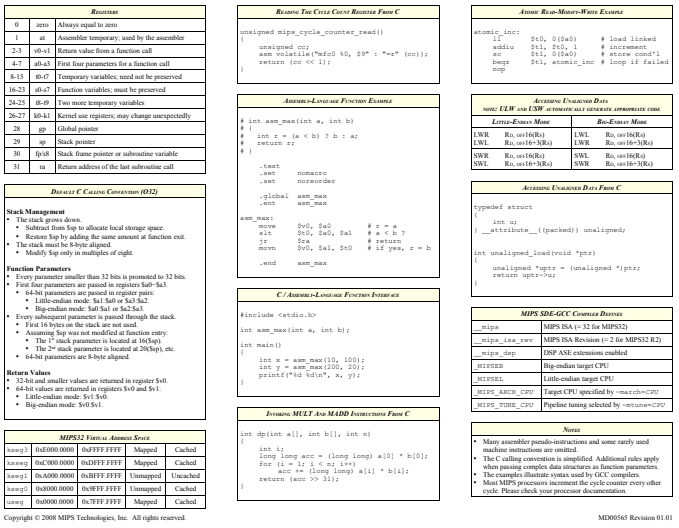
LB	Rs, const16(Rs)	Rs = sim8(Rs + const16)*
LBU	Rs, const16(Rs)	Rs = sim8(Rs + const16)#
LH	Rs, const16(Rs)	Rs = sim16(Rs + const16)*
LHU	Rs, const16(Rs)	Rs = sim16(Rs + const16)#
LW	Rs, const16(Rs)	Rs = sim32(Rs + const16)
LWL	Rs, const16(Rs)	Rs = Load_WordLeft(Rs + const16)
LWR	Rs, const16(Rs)	Rs = Load_WordRight(Rs + const16)
SB	Rs, const16(Rt)	sim8(Rt + const16) = Rs_{7:0}
SH	Rs, const16(Rt)	sim16(Rt + const16) = Rs_{15:0}
SW	Rs, const16(Rt)	sim32(Rt + const16) = Rs
SWL	Rs, const16(Rt)	Store_WordLeft(Rt + const16*, Rs)
SWR	Rs, const16(Rt)	Store_WordRight(Rt + const16*, Rs)
LLW	Rs, const16(Rt)	Rs = invalidate_sim32(Rt + const16*)
LSW	Rs, const16(Rt)	invalidate_sim32(Rt + const16*) = Rs

Atomic Read-Modify-Write Operations

LL	Rs, const16(Rt)	Rs = sim32(Rt + const16*); link
SC	Rs, const16(Rt)	si Atomic, sim32(Rt + const16*) = Rs; Rs = Atomic ? 1 : 0

MD00565 Revision 01.01

Zoom. . . Niveau 2



Registers

0	zero	Always equal to zero
1	at	Assembler temporary; used by the assembler
2-3	v0-v1	Return value from a function call
4-7	a0-a3	First four parameters for a function call
8-15	t0-t7	Temporary variables; need not be preserved
16-23	s0-s7	Function variables; must be preserved
24-25	t8-t9	Two more temporary variables
26-27	k0-k1	Kernel use registers; may change unexpectedly
28	gp	Global pointer
29	sp	Stack pointer
30	fp/s8	Stack frame pointer or subroutine variable
31	ra	Return address of the last subroutine call

Default C Calling Conventions (O32)

Stack Management
- The stack grows down.
  - Subtract from $sp to allocate local storage space.
  - Restore $sp by adding the same amount at function exit.
- The stack must be 8-byte aligned.
  - Modify $sp only in multiples of eight.

Function Parameters
- Every parameter smaller than 32 bits is promoted to 32 bits.
- First four parameters are passed in registers $a0–$a3.
  - 64-bit parameters are passed in register pairs:
    - Little-endian mode: $a1-$a0 or $a3-$a2.
    - Big-endian mode: $a0-$a1 or $a2-$a3.
- Every subsequent parameter is passed through the stack.
  - First 16 bytes on the stack are not used.
  - Assuming $sp was not modified at function entry:
    - The 1st stack parameter is located at 16(Sap).
    - The 2nd stack parameter is located at 20(Sap), etc.
  - 64-bit parameters are 8-byte aligned.

Return Values
- 32-bit and smaller values are returned in register $v0.
- 64-bit values are returned in registers $v0 and $v1:
  - Little-endian mode: $v1 $v0.
  - Big-endian mode: $v0-$v1.

MIPS32 Virtual Address Space

kseg2	0xE000 0000	0xFFFF FFFF	Mapped	Cached
ksseg	0xC000 0000	0xDFFF FFFF	Mapped	Cached
kseg1	0xA000 0000	0xBFFF FFFF	Unmapped	Uncached
kseg0	0x8000 0000	0x9FFF FFFF	Unmapped	Cached
useg	0x0000 0000	0x7FFF FFFF	Mapped	Cached

Reading The Cycle Count Register From C

```
unsigned mips_cycle_counter_read()
{
    unsigned cc;
    asm volatile("mfc0 %0, $9" : "=r" (cc));
    return (cc << 1);
}
```

Assembly-Language Function Example

```
# int asm_max(int a, int b)
# {
#   int r = (a < b) ? b : a;
#   return r;
# }

    .text
    .set    nomacro
    .set    noreorder

    .global asm_max
    .ent    asm_max

asm_max:
    move    $v0, $a0        # r = a
    slt     $t0, $a0, $a1   # a < b ?
    jr      $ra             # return
    movn    $v0, $a1, $t0   # if yes, r = b

    .end    asm_max
```

C / Assembly-Language Function Interface

```
#include <stdio.h>

int asm_max(int a, int b);

int main()
{
    int x = asm_max(10, 100);
    int y = asm_max(200, 20);
    printf("%d %d\n", x, y);
}
```

Invoking MULT And MADD Instructions From C

```
int dp(int a[], int b[], int n)
{
    int i;
    long long acc = (long long) a[0] * b[0];
    for (i = 1; i < n; i++)
        acc += (long long) a[i] * b[i];
    return (acc >> 31);
}
```

Atomic Read-Modify-Write Example

```
atomic_inc:
    ll      $t0, 0($a0)     # load linked
    addiu   $t1, $t0, 1     # increment
    sc      $t1, 0($a0)     # store cond'l
    beqz    $t1, atomic_inc # loop if failed
    nop
```

Accessing Unaligned Data note: ULW and USW automatically generate appropriate code

Little-Endian Mode	Big-Endian Mode
LWR R0, 0x16(R0)	LWL R0, 0x16(R0)
LWL R0, 0x16+3(R0)	LWR R0, 0x16+3(R0)
SWR R0, 0x16(R0)	SWL R0, 0x16(R0)
SWL R0, 0x16+3(R0)	SWR R0, 0x16+3(R0)

Accessing Unaligned Data From C

```
typedef struct
{
    int u;
} __attribute__((packed)) unaligned;

int unaligned_load(void *ptr)
{
    unaligned *uptr = (unaligned *)ptr;
    return uptr->u;
}
```

MIPS SDE-GCC Compiler Defines

__mips	MIPS ISA (= 32 for MIPS32)
__mips_isa_rev	MIPS ISA Revision (= 2 for MIPS32 R2)
__mips_dsp	DSP ASE extensions enabled
__MIPSEB	Big-endian target CPU
__MIPSEL	Little-endian target CPU
__MIPS_ARCH_CPU	Target CPU specified by -march=CPU
__MIPS_TUNE_CPU	Pipeline tuning selected by -mtune=CPU

Notes
- Many assembler pseudo-instructions and some rarely used machine instructions are omitted.
- The C calling convention is simplified. Additional rules apply when passing complex data structures as function parameters.
- The examples illustrate syntax used by GCC compilers.
- Most MIPS processors increment the cycle counter every other cycle. Please check your processor documentation.

MD00565 Revision 01.01

Zoom. . . Niveau 2

Format des instructions ?

- Longueur fixe ? longueur variable ?
- Nombre d'opérandes par instruction ?
- Nombre d'opérandes mémoire par instruction ?

RISC : fixe, 3 opérandes, 0 mémoire (sauf lect. et écr.)

CISC : tout le reste (IA-32 par exemple)

⇝ RISC cycle d'exécution simple et prévisible

⇝ CISC instructions plus complexes à disposition

Zoom. . . Niveau 2

Mode d'adressage ? Comment accéder aux opérandes ?

Mode registre : l'opérande est un registre (valeur contenue dans registre désigné)

EX. en supposant `ADD` : 0100 en LM sur instructions 16 bits

- `ADD DR, SR1, SR2` effectue `DR <- SR1 + SR2`
  0100 0111 0001 0010
  range dans R7 la somme des contenus de R1 et R2

Zoom. . . Niveau 2

Mode d'adressage ? Comment accéder aux opérandes ?

Mode registre : l'opérande est un registre (valeur dans registre désigné)

Mode immédiat : l'opérande est directement la valeur

EX. en supposant `ADDI` : 0101 en LM sur instructions 16 bits

- `ADDI DR, SR1, IMM` effectue `DR <- SR1 + IMM`
  0101 0111 0001 0010
  range dans R7 le contenu de R1 plus 2

Zoom. . . Niveau 2

Mode d'adressage ? Comment accéder aux opérandes ?

Mode registre : l'opérande est un registre (valeur dans registre désigné)

Mode immédiat : l'opérande est directement la valeur

Mode direct : adresse dans l'instruction

EX. en supposant `STRD` : 1101 en LM sur instructions 16 bits

- `STRD SR1, ADR` range le contenu de `SR1` à l'adresse `ADR`
  1101 0111 0001 0010
  range à l'adresse 18 le contenu de R7

Zoom. . . Niveau 3

Pénible si on programme pour plusieurs architectures. . .

Système d'exploitation (OS) : *abstractions et outils pour gérer archi et matériel*

- Uniformisation commandes, accès — Visible au programmeur
- Spécialisation par architecture — Invisible au programmeur

Ex. Linux
même interface pour les versions AMD64, ARM, PowerPC, IA-32 (x86). . .

Zoom... Niveau 4

Jusqu'ici : que des 0 et des 1 on veut un peu de confort !

Langage d'assemblage : langage de niveau intermédiaire

- Très très simple (au sens primitif)
- Traduit vers OS + LM à l'aide d'un assembleur

Zoom... Niveau 5

Langages de haut niveau (Ocaml, C++, etc.)

- Fonctions et opérations complexes
- Traduits vers LA + OS à l'aide de compilateurs

Opérations complexes décomposées en instructions...

Exemple Niveaux 5 -> 1

Programme en C, *langage de haut niveau* :

```
#include <stdio.h>
char car;
int main(void) {
  printf("Hi!\n");                    // appel à une primitive de l'OS
  printf("Entrez un caractere...\n");
  car = getchar();                    // appel à une primitive de l'OS
  printf("Vous avez entre : ");
  putchar(car);                       // appel à une primitive de l'OS
  putchar('\n');
  printf("Bye!\n");
  return(0);
}
```


Exemple Niveaux 5 -> 1

```
        LEA R0,msg0     ; charge l'adresse effective désignée par msg0 dans R
        TRAP x22        ; affiche la chaine pointée par R0
        LEA R0,msg1     ;
        TRAP x22        ; affiche la chaine à l'adresse msg1
        TRAP x20        ; lit un caractère et le place dans R0
        ST R0,car       ; stocke le caractère lu à l'adresse car
        LEA R0,msg2     ;
        TRAP x22        ; affiche la chaine à l'adresse msg2
        LD R0,car       ; charge le caractère stocké à l'adresse car dans R0
        TRAP x21        ; affiche le caractère qui a été lu
        LEA R0,ret      ;
        TRAP x22        ; affiche un retour à la ligne
        LEA R0,msg3     ;
        TRAP x22        ; affiche la chaine à l'adresse msg3
        TRAP x25        ; termine le programme (rend la main à l'OS)
car:    .BLKW #1        ; case mémoire pour stocker un caractère lu
msg0:   .STRINGZ "Hi!\n"
msg1:   .STRINGZ "Entrez un caractere...\n"
msg2:   .STRINGZ "Vous avez entre : "
```

Exemple — Niveaux 5 -> 1

Traduit en *langage machine* par l'*assembleur* :

```
E00F F022 E012 F022 F020 3009 E026 F022 2006 ...
```

On a la correspondance suivante pour chaque instruction :

```
langage              langage    langage
machine              machine    d'assemblage
                     hexa

1110 0000 0000 1111   xE00F    LEA R0,msg0
1111 0000 0010 0010   xF022    TRAP x22
1110 0000 0001 0010   xE012    LEA R0,msg1
1111 0000 0010 0010   xF022    TRAP x22
1111 0000 0010 0000   xF020    TRAP x20
0011 0000 0000 1001   x3009    ST R0,car
1110 0000 0010 0110   xE026    LEA R0,msg2
1111 0000 0010 0010   xF022    TRAP x22
0010 0000 0000 0110   x2006    LD R0,car
...
```


Ordinateurs... — Problématiques

Performances ?

- Général ?
- Spécialisé ?
  - Calcul scientifique
  - BD
  - Traitement du signal (DSP)
  - Graphique (lourd mais simple)

Ordinateurs... — Problématiques

Performances ?

OPS ?
Comm. ?
Efficacité énergétique ?

En fonction de gravure :

	90nm	65nm	45nm	32nm	22nm
GOPS	2	14	77	461	2458
kbps	384	2304	13824	82944	497664
mW max			100		

Ordinateurs... — Problématiques

Calcul / Mémoire / Com.

Métronome : horloge

Lect. instr. Lect. op. Exécution Rangement

Ordinateurs. . . Formule fondamentale

Calcul / Mémoire / Com.

Métronome : horloge

$$T_{\text{ex}} = NI \times CPI \times T_c = \frac{NI}{IPC \times F}$$

- NI : nombre d'instructions
- CPI : cycles par instruction = $1/IPC$
- T_c : Temps par cycle = $1/F$
- F : fréquence d'horloge

Ordinateurs. . . Formule fondamentale

Calcul / Mémoire / Com.

Métronome : horloge

$$T_{\text{ex}} = NI \times CPI \times T_c = \frac{NI}{IPC \times F}$$

1987 → 2004

Horloge : +25%/an
Calcul : +60%/an

Ordinateurs. . . Problématiques

Calcul / Mémoire / Com.

Métronome : horloge

	RAM	Secondaire	Perf. calcul
Bande passante :	+20%/an	+40%/an	
Latence :	+6%/an	+7%/an	+60%/an
Capacité :	+60%/an		

Ordinateurs. . . techno ?

Engrenages

Tubes à vides

Transistors

- nMOS, pMOS — semi-conducteurs
- CMOS — complementary-

Gravure :

	1985	. . .	2002	2006	2008	2010	2012
nm	1000	. . .	90	65	45	32	22

Ordinateurs. . .

« Loi » de Moore : 2 ans → × 2 transistors

1985 :	80386	→	275000
1989 :	80486	→	1000000
1995 :	Prentium Pro	→	5500000
2003 :	K8	→	100000000
2011 :	core i7	→	995000000
2014 :	POWER8	→	4200000000

Fonctionnement 1V ⇝ courant. . . chaleur. . .

Ordinateurs. . .

Densité puissance : exponentielle (monoproc.)

Pentium Pro : ~10W/cm^2

Pentium 4 : ~50W/cm^2

Cœur centrale : ~200W/cm^2 oups. . .

⇝ architectures parallèles

Ordinateurs. . .

Densité puissance : exponentielle (monoproc.)

Puissance dissipée :

$$P_d = P_s + \alpha \sum_i C_i V^2 F$$

- P_s : puissance statique
- α : pourcentage de composants actifs
- C_i : capacité
- V : tension
- F : fréquence d'horloge

Ordinateurs. . .

Densité puissance : exponentielle (monoproc.)

Puissance dissipée :

$$P_d = P_s + \alpha \sum_i C_i V^2 F$$

Linéairement proportionnel à F

Ordinateurs. . .

Densité puissance : exponentielle (monoproc.)

Puissance dissipée :

$$P_d = P_s + \alpha \sum_i C_i V^2 F$$

Baisser P_s ?

Non : fuites

Ordinateurs. . .

Densité puissance : exponentielle (monoproc.)

Puissance dissipée :

$$P_d = P_s + \alpha \sum_i C_i V^2 F$$

Baisser V ?

Non : seuil techno

Ordinateurs. . .

Densité puissance : exponentielle (monoproc.)

Puissance dissipée :

$$P_d = P_s + \alpha \sum_i C_i V^2 F$$

Baisser C_i ?

Bof : bond techno mais augmentation nb. transistors

Ordinateurs. . .

Densité puissance : exponentielle (monoproc.)

Puissance dissipée :

$$P_d = P_s + \alpha \sum_i C_i V^2 F$$

Linéairement proportionnel à F

Reste : difficile d'y toucher

$\rightsquigarrow$ limite horloge : ~4GHz (depuis 2000)

Gravure plus fine $\rightsquigarrow$ densité !

	45nm	22nm	11nm
Util. F_{45}	100%	40%	20%
Util. F_{max}	100%	25%	10%

Ordinateurs. . .

Calcul / Mémoire / Com.

Métronome : horloge

$$T_{\text{ex}} = NI \times CPI \times T_c = \frac{NI}{IPC \times F}$$

Augmenter F ?

Non : puissance. . .

Ordinateurs. . .

Calcul / Mémoire / Com.

Métronome : horloge

$$T_{\text{ex}} = NI \times CPI \times T_c = \frac{NI}{IPC \times F}$$

Augmenter IPC ?

- Pipelines (et super-pipelines)
- Super-scalaires

Ordinateurs. . .

Augmenter IPC ? Pipelines

Séquentiel :

Lec I	Lec O	Exé	Ran	Lec I	Lec O	Exé	Ran

Pipeline :

Lec I	Lec O	Exé	Ran				
	Lec I	Lec O	Exé	Ran			
		Lec I	Lec O	Exé	Ran		
			Lec I	Lec O	Exé	Ran	
				Lec I	Lec O	Exé	Ran

Ordinateurs. . .

Augmenter IPC ? Pipelines

Séquentiel :

Lec I	Lec O	Exé	Ran	Lec I	Lec O	Exé	Ran

Superscalaires :

Lec I	Lec O	Exé	Ran
Lec I	Lec O	Exé	Ran
	Lec I	Lec O	Exé	Ran
	Lec I	Lec O	Exé	Ran

Quand possible

Ordinateurs. . .

Calcul / Mémoire / Com.

Métronome : horloge

$$T_{\text{ex}} = NI \times CPI \times T_c = \frac{NI}{IPC \times F}$$

Augmenter IPC ?

$\rightsquigarrow$ Mouais : Pentium Pro $\rightarrow$ 3, i7 4$^{\text{e}}$ gén. $\rightarrow$ 4

Ordinateurs. . .

Calcul / Mémoire / Com.

Métronome : horloge

$$T_{\text{ex}} = NI \times CPI \times T_c = \frac{NI}{IPC \times F}$$

Diminuer NI ?

Compilateurs +

- SIMD (→ vect. 512 bits)
- SIMT (GPU)
- Archi. parallèles

Ce cours. . .

Monoprocesseur assez simple

mais entièrement. . .