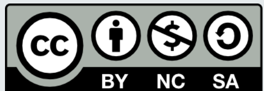
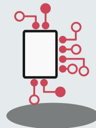
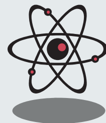
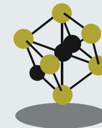
Chapitre 5

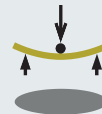
# Langage d'assemblage



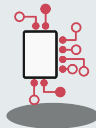
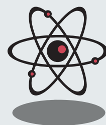


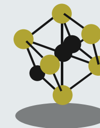
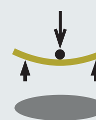






# INSTRUCTION ASSEMBLEUR

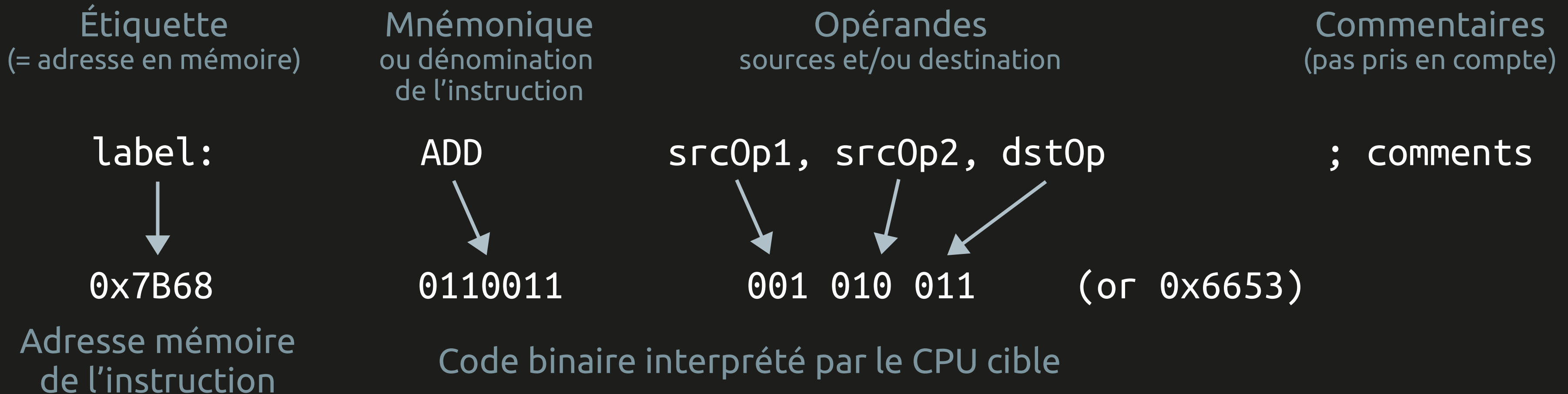



## Assembleur et code binaire

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Un **langage d'assemblage** ou **assembleur** ou **ASM** est un **langage de programmation bas niveau** représentant, sous forme lisible pour un être humain, le code binaire exécutable par un processeur (ou code machine).

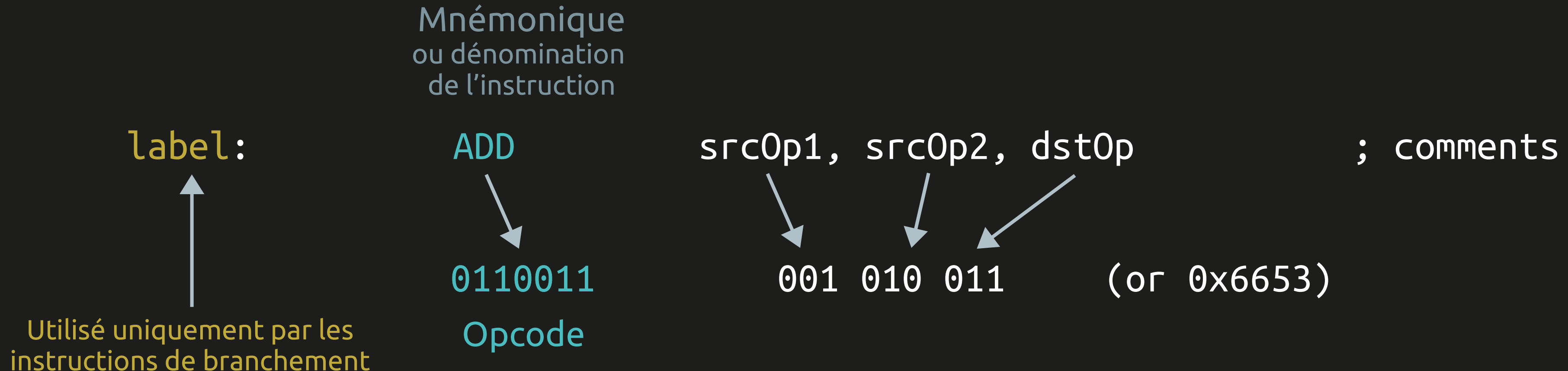
Prenons l'exemple d'une instruction assembleur élémentaire raccrochée à aucune architecture connue :



En général, à tout champ d'une instruction assembleur correspond un champ dans le code binaire équivalent (sauf label et commentaires).

Ce code binaire ne peut être compris et interprété que par le CPU cible.

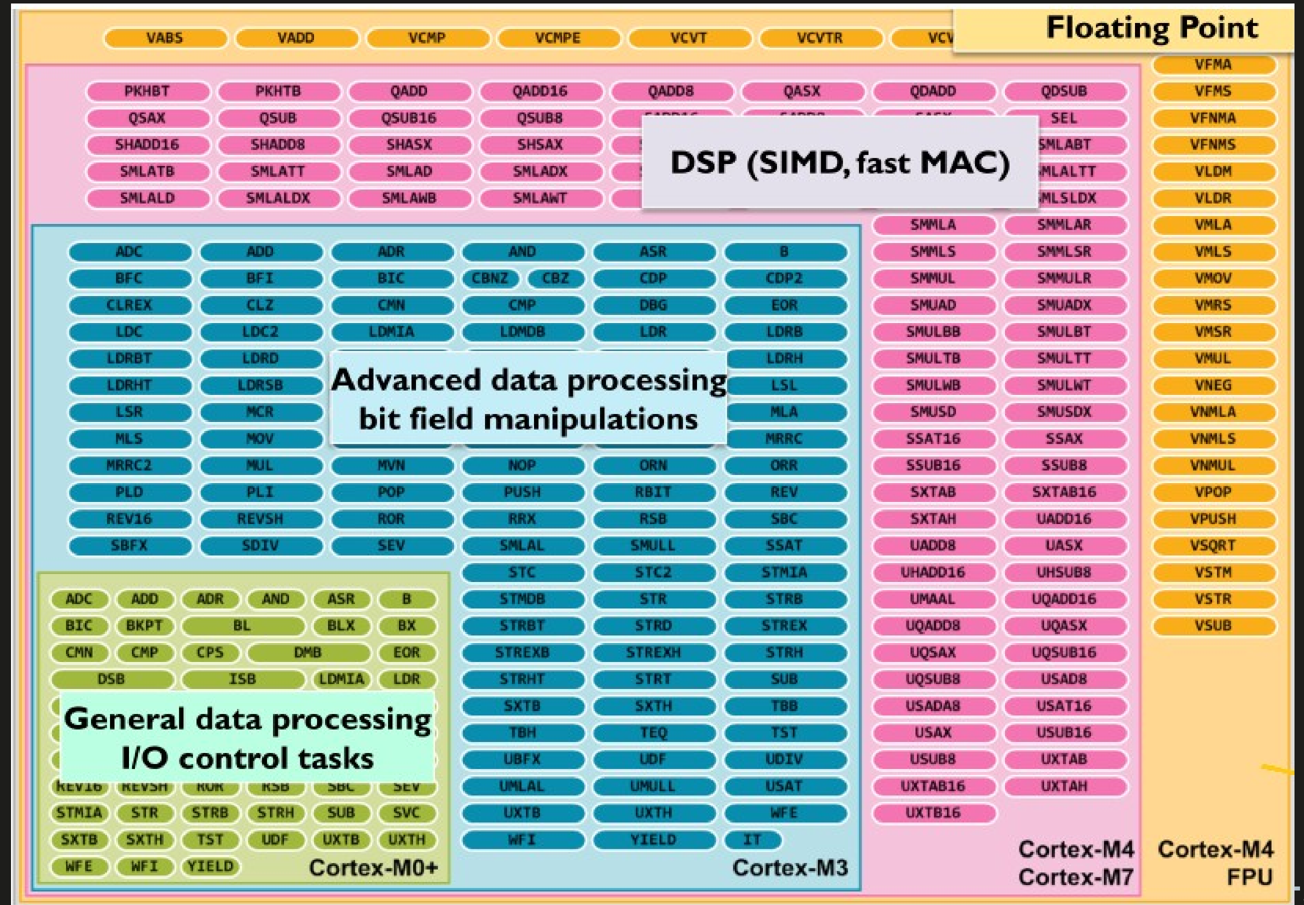
Rappelons également que comme tout langage de programmation, la syntaxe ne fait appel qu'à des références symboliques. La résolution des symboles étant gérée à l'édition des liens (exemple des labels, des adresses mémoire des variables …) :



L’assembleur est le langage de programmation **le moins universel au monde**.

Il existe autant de langage d’assemblage que de familles de CPU.

Voici par exemple le jeu d’instructions supporté par la famille ARM Cortex-M.



ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Observons les principaux acteurs dans le domaine des CPUs. Chaque fondeur présenté ci-dessous propose une voire plusieurs architectures de CPU qui lui sont propres et possédant donc les jeux d'instructions associés :

## GPP CPU architectures

Intel (IA-32 et Intel 64), AMD (x86 et AMD64), IBM (PowerPC), Renesas (RX CPU), Zilog (Z80), Motorola (6800 et 68000) …

## Embedded CPU architectures (MCU, DSP, SoC)

ARM (Cortex-M, -R, -A), MIPS (Rx000), Intel (Atom, 8051), Renesas, Texas Instrument (MSPxxx, C2xxx, C5xxx, C6xxx), Microchip (PICxx) , Atmel (AVR), Apple/IBM/Freescale (PowerPC) …

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Tout CPU est capable de décoder puis d'exécuter un jeu d'instructions qui lui est propre (**ISA ou I*nstruction Set Architecture***). Celles-ci peuvent être classées en trois familles :

## Calcul et comparaison

Opérations arithmétiques et logiques (en C : +, -, *, /, &, |, …) et opérations de comparaison (en C : >, <=, !=, == …). Les formats entiers courts seront toujours supportés nativement. En fonction de l'architecture du CPU, les formats entiers long (16-bit et plus) voire flottants peuvent l'être également.

## Manipulation de données

Déplacement de données dans l'architecture matérielle (CPU → CPU, CPU → mémoire ou mémoire → CPU)

## Contrôle programme

Saut en mémoire programme (saut dans le code). Par exemple en langage C : if, else if, else, switch, for, while, do while, appels de procédure. Ces sauts peuvent être rendus conditionnels à l'aide d'opérations arithmétiques et logiques ou de comparaisons.

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Certaines architectures, comme les architectures compatibles x86-64 (Intel et AMD), possèdent des familles spécialisées :

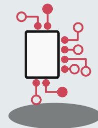
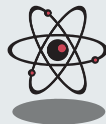
## String manipulation

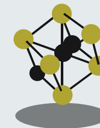
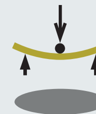
Manipulation au niveau assembleur de chaînes de caractères.

## Divers

Arithmétique lourde (sinus, cosinus…), opérations vectorielles (produit vectoriel, produit scalaire…) …

# MODES D’ADRESSAGE

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Tout bon CPU est capable de déplacer des données dans l'architecture du processeur :

- **registre (CPU) vers mémoire**
- **registre (CPU) vers registre (CPU)**
- **mémoire vers registre (CPU)**

On peut souligner que le déplacement mémoire vers mémoire n'est pas implémenté. D'une part il n'a quasiment aucun intérêt, d'autre part on peut toujours le réaliser par un déplacement mémoire → CPU suivi d'un déplacement CPU → mémoire.

Si nous souhaitons réaliser des transferts mémoire → mémoire sans passer par le cœur, les périphériques spécialisés de type DMA (*Direct Memory Access*) peuvent s'en charger (si le processeur en est pourvu).

ENSI CAEN
ÉCOLE PUBLIQUE D’INGÉNIEURS
CENTRE DE RECHERCHE

Dans une instruction, il existe différentes manières de spécifier l’emplacement mémoire d’un opérande : il s’agit des différents **modes d’adressage**.

Concentrons nous sur les plus répandus.

## Adressage immédiat

→ `mov     $0x1A2F, %bx`

La valeur de l’opérande (une constante) est écrite en dur dans le code binaire de l’instruction.

✓ pas besoin d’accès à la mémoire

✗ on ne peut manipuler que des constantes

✗ les valeurs de la constante sont limitées par le nombre de bits du champ de l’opérande

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

## Adressage registre

→ `mov %ax, %bx`

Le code binaire de l'instruction contient le numéro du registre à manipuler.

✓ pas besoin d'accès à la mémoire

✓ le mode de loin le plus répandu, presque 100 % des instructions des ISA LOAD/STORE utilisent ce mode

✗ le nombre de données accessibles dépend du nombre de registres du CPU

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

## Adressage direct

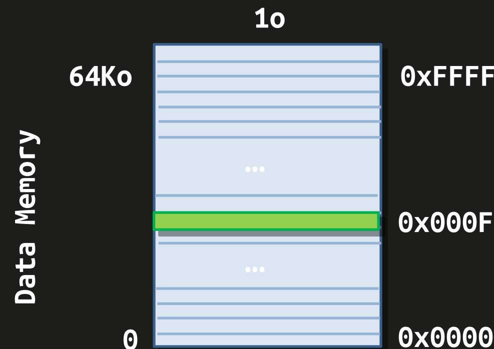
→ `mov (0x000F), %bl`

Le code binaire de l'instruction contient l'adresse de la case mémoire.

✓ pas besoin de calculer/récupérer une adresse (elle est écrite en dur dans le code binaire de l'instruction)

✗ ne fonctionne qu'avec des données dont l'adresse est connue à la compilation

✓ … ce qui arrive très souvent (variables globales, statiques)



ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

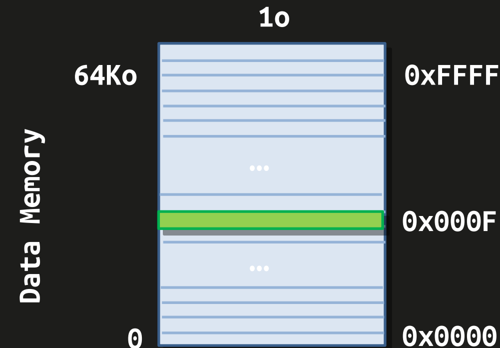
## Adressage indirect

```
→ mov     0x000F, %ax
→ mov     (%ax),  %bl
```

Le code binaire de l'instruction contient le numéro d'un registre, qui lui contient l'adresse de la donnée (en gros, le registre est un pointeur)

✓ l'instruction ne contient pas l'adresse complète, mais seulement le numéro d'un registre

✓ dans les boucles, il suffit d'incrémenter le même registre pour passer d'une case à la suivante



ENSI CAEN
ÉCOLE PUBLIQUE D’INGÉNIEURS
CENTRE DE RECHERCHE

## Adressage indexé

Non vu en cours (les registres SI et DI du x86 utilisent ce mode d’adressage).

L’instruction contient une adresse réelle (en dur dans le code binaire) et un numéro de registre qui lui contient un offset (+/-).

Par exemple on pourrait stocker l’adresse de départ d’une chaîne de caractère et le registre pourrait contenir le n° de case.

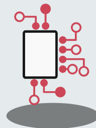
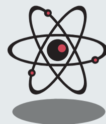
## Modes d'adressage disponibles pour l'instruction MOV

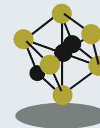
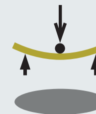
Vous pouvez observer ci-dessous la totalité des modes d'adressage supportés sur architecture Intel 64 actuelle concernant l'instruction `MOV` "seule". La gestion de nombreux modes d'adressage implique une complexité accrue des unités de décodage et d'exécution.

### MOV—Move

| Opcode | Instruction | Op/ En | 64-Bit Mode | Compat/ Leg Mode | Description |
|---|---|---|---|---|---|
| 88 /r | MOV r/m8, r8 | MR | Valid | Valid | Move r8 to r/m8. |
| REX + 88 /r | MOV r/m8[1], r8[1] | MR | Valid | N.E. | Move r8 to r/m8. |
| 89 /r | MOV r/m16, r16 | MR | Valid | Valid | Move r16 to r/m16. |
| 89 /r | MOV r/m32, r32 | MR | Valid | Valid | Move r32 to r/m32. |
| REX.W + 89 /r | MOV r/m64, r64 | MR | Valid | N.E. | Move r64 to r/m64. |
| 8A /r | MOV r8, r/m8 | RM | Valid | Valid | Move r/m8 to r8. |
| REX + 8A /r | MOV r8[1], r/m8[1] | RM | Valid | N.E. | Move r/m8 to r8. |
| 8B /r | MOV r16, r/m16 | RM | Valid | Valid | Move r/m16 to r16. |
| 8B /r | MOV r32, r/m32 | RM | Valid | Valid | Move r/m32 to r32. |
| REX.W + 8B /r | MOV r64, r/m64 | RM | Valid | N.E. | Move r/m64 to r64. |
| 8C /r | MOV r/m16, Sreg[2] | MR | Valid | Valid | Move segment register to r/m16. |
| 8C /r | MOV r16/r32/m16, Sreg[2] | MR | Valid | Valid | Move zero extended 16-bit segment register to r16/r32/m16. |
| REX.W + 8C /r | MOV r64/m16, Sreg[2] | MR | Valid | Valid | Move zero extended 16-bit segment register to r64/m16. |
| 8E /r | MOV Sreg, r/m16[2] | RM | Valid | Valid | Move r/m16 to segment register. |
| REX.W + 8E /r | MOV Sreg, r/m64[2] | RM | Valid | Valid | Move lower 16 bits of r/m64 to segment register. |
| A0 | MOV AL, moffs8[3] | FD | Valid | Valid | Move byte at (seg:offset) to AL. |
| REX.W + A0 | MOV AL, moffs8[3] | FD | Valid | N.E. | Move byte at (offset) to AL. |
| A1 | MOV AX, moffs16[3] | FD | Valid | Valid | Move word at (seg:offset) to AX. |
| A1 | MOV EAX, moffs32[3] | FD | Valid | Valid | Move doubleword at (seg:offset) to EAX. |
| REX.W + A1 | MOV RAX, moffs64[3] | FD | Valid | N.E. | Move quadword at (offset) to RAX. |
| A2 | MOV moffs8, AL | TD | Valid | Valid | Move AL to (seg:offset). |
| REX.W + A2 | MOV moffs8[1], AL | TD | Valid | N.E. | Move AL to (offset). |
| A3 | MOV moffs16[3], AX | TD | Valid | Valid | Move AX to (seg:offset). |
| A3 | MOV moffs32[3], EAX | TD | Valid | Valid | Move EAX to (seg:offset). |
| REX.W + A3 | MOV moffs64[3], RAX | TD | Valid | N.E. | Move RAX to (offset). |
| B0+ rb ib | MOV r8, imm8 | OI | Valid | Valid | Move imm8 to r8. |
| REX + B0+ rb ib | MOV r8[1], imm8 | OI | Valid | N.E. | Move imm8 to r8. |
| B8+ rw iw | MOV r16, imm16 | OI | Valid | Valid | Move imm16 to r16. |
| B8+ rd id | MOV r32, imm32 | OI | Valid | Valid | Move imm32 to r32. |
| REX.W + B8+ rd io | MOV r64, imm64 | OI | Valid | N.E. | Move imm64 to r64. |
| C6 /0 ib | MOV r/m8, imm8 | MI | Valid | Valid | Move imm8 to r/m8. |
| REX + C6 /0 ib | MOV r/m8[1], imm8 | MI | Valid | N.E. | Move imm8 to r/m8. |
| C7 /0 iw | MOV r/m16, imm16 | MI | Valid | Valid | Move imm16 to r/m16. |
| C7 /0 id | MOV r/m32, imm32 | MI | Valid | Valid | Move imm32 to r/m32. |
| REX.W + C7 /0 id | MOV r/m64, imm32 | MI | Valid | N.E. | Move imm32 sign extended to 64-bits to r/m64. |

# RISC VS CISC

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Les jeux d'instructions et CPU associés peuvent être classés en 2 grandes familles :

- **RISC** pour *Reduced Instruction Set Computer*
- **CISC** pour *Complex Instruction Set Computer*

Les architectures RISC n'implémentent en général que des instructions élémentaires (CPUs ARM, MIPS, 8051, PIC18 …).

À l'inverse, les architectures CISC (CPUs x86-64, 68xxx …) implémentent nativement au niveau assembleur des traitements pouvant être très complexes (division, opérations vectorielles, opérations sur des chaînes de caractères …).

En 2012, la frontière entre ces deux familles est de plus en plus fine. Par exemple, le jeu d'instructions des processeurs spécialisés DSP RISC-like TMS320C66xx de TI compte 323 instructions. Néanmoins, les architectures compatibles x86-64 sont des architectures CISC. Nous allons rapidement comprendre pourquoi.

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

## Inconvénients architecture RISC

- Empreinte mémoire programme élevée, donc moins d'instructions contenues en cache et mémoire principale.

## Avantages architecture RISC

- Architecture du CPU moins complexe (mécanismes d'accélération matériels, décodeurs, unités d'exécution …), donc moins cher
- En général, tailles instructions fixes et souvent exécution en un ou deux cycles CPU.
- Jeu d'instructions plus simple à appréhender pour le développeur et donc le compilateur. Jeu d'instructions très bien géré par les chaînes de compilations (mécanismes d'optimisation).
- Beaucoup d'architectures RISC récentes, travaillent avec de nombreux registres de travail généralistes, facilite le travail du compilateur.

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

## Avantages architecture CISC

- Empreinte mémoire programme faible, donc plus d'instructions contenues en cache. Néanmoins sur CPU CISC, en moyenne près de 80% des instructions compilées sont de types RISC.
- Compatibles x86-64, rétrocompatibilité des applications développées sur anciennes architectures.

## Inconvénients architecture CISC

- Architecture CPU complexe (mécanismes d'accélération matériels, décodeurs, Execution Units ...), donc moins de place pour le cache.
- Jeu d'instructions mal géré par les chaînes de compilation (mécanismes d'optimisation)

## Exemple de CPU RISC : Intel 8051



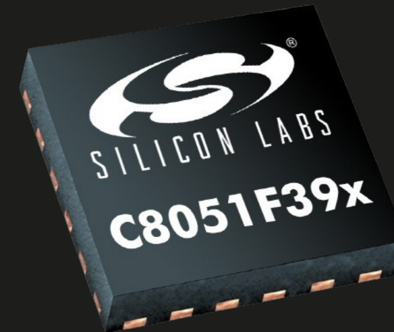
Observons le jeu d'instructions complet d'un CPU RISC 8051 proposé par Intel en 1980.

En 2012, cette famille de CPU, même si elle reste très ancienne, est toujours extrêmement répandue et intégrée dans de nombreux MCUs ou ASICs (licence libre).

Prenons quelques exemples de fondeurs les utilisant : NXP, silabs, Atmel …

8051 Intel CPU (1980)
*CPU only, not a MCU*

Silabs MCU (2012)
*MCU with a 8051 CPU*



## Exemple de CPU RISC : Intel 8051

# RISC CPU

## 8051 Instruction set

| | | | |
|---|---|---|---|
| **ACALL** | *Absolute Call* | **MOV** | *Move Memory* |
| **ADD, ADDC** | *Add Accumulator (With Carry)* | **MOVC** | *Move Code Memory* |
| **AJMP** | *Absolute Jump* | **MOVX** | *Move Extended Memory* |
| **ANL** | *Bitwise AND* | **MUL** | *Multiply Accumulator by B* |
| **CJNE** | *Compare and Jump if Not Equal* | **NOP** | *No Operation* |
| **CLR** | *Clear Register* | **ORL** | *Bitwise OR* |
| **CPL** | *Complement Register* | **POP** | *Pop Value From Stack* |
| **DA** | *Decimal Adjust* | **PUSH** | *Push Value Onto Stack* |
| **DEC** | *Decrement Register* | **RET** | *Return From Subroutine* |
| **DIV** | *Divide Accumulator by B* | **RETI** | *Return From Interrupt* |
| **DJNZ** | *Decrement Register and Jump if Not Zero* | **RL** | *Rotate Accumulator Left* |
| **INC** | *Increment Register* | **RLC** | *Rotate Accumulator Left Through Carry* |
| **JB** | *Jump if Bit Set* | **RR** | *Rotate Accumulator Right* |
| **JBC** | *Jump if Bit Set and Clear Bit* | **RRC** | *Rotate Accumulator Right Through Carry* |
| **JC** | *Jump if Carry Set* | **SETB** | *Set Bit* |
| **JMP** | *Jump to Address* | **SJMP** | *Short Jump* |
| **JNB** | *Jump if Bit Not Set* | **SUBB** | *Subtract From Accumulator With Borrow* |
| **JNC** | *Jump if Carry Not Set* | **SWAP** | *Swap Accumulator Nibbles* |
| **JNZ** | *Jump if Accumulator Not Zero* | **XCH** | *Exchange Bytes* |
| **JZ** | *Jump if Accumulator Zero* | **XCHD** | *Exchange Digits* |
| **LCALL** | *Long Call* | **XRL** | *Bitwise Exclusive OR* |
| **LJMP** | *Long Jump* | | |

## Exemple de CPU CISC : Intel 8086

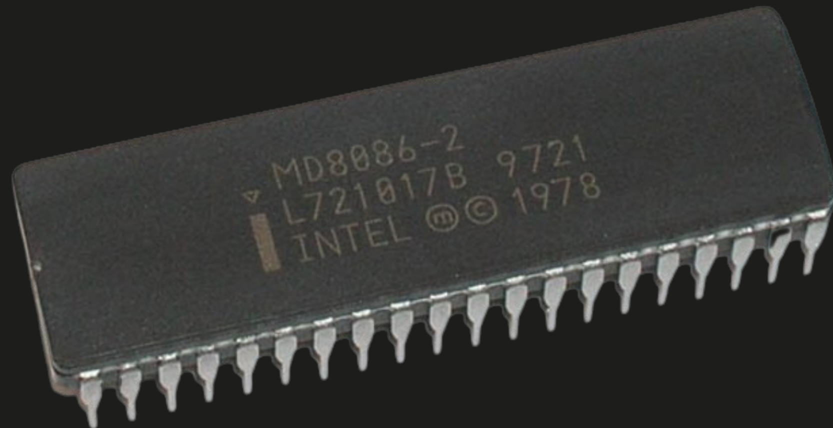
ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Observons le jeu d'instructions complet d'un CPU 16-bit CISC 8086 proposé par Intel en 1978.

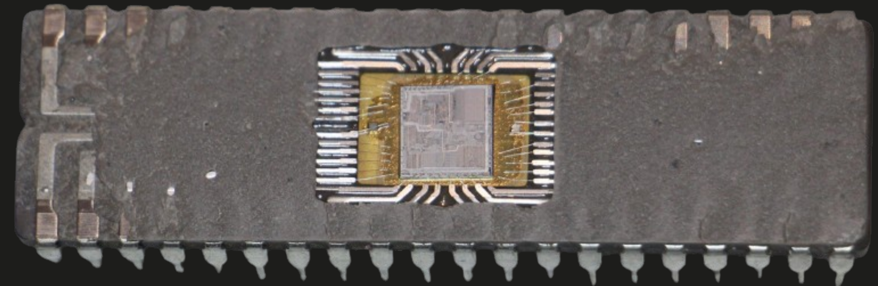
Il s'agit du premier processeur de la famille x86.

Les processeurs modernes de la famille Intel Core-$i_x$ sont toujours capables d'exécuter le jeu d'instructions d'un 8086. Bien sûr, la réciproque n'est pas vraie.

8086 Intel CPU (1978)




8086 Intel CPU (1978)
*(the same one, just open for science)*



## Exemple de CPU CISC : Intel 8086

# CISC CPU

# 8086 Instruction set (1/2)

| | |
|---|---|
| **AAA** | *ASCII adjust AL after addition* |
| **AAD** | *ASCII adjust AX before division* |
| **AAM** | *ASCII adjust AX after multiplication* |
| **AAS** | *ASCII adjust AL after subtraction* |
| **ADC** | *Add with carry* |
| **ADD** | *Add* |
| **AND** | *Logical AND* |
| **CALL** | *Call procedure* |
| **CBW** | *Convert byte to word* |
| **CLC** | *Clear carry flag* |
| **CLD** | *Clear direction flag* |
| **CLI** | *Clear interrupt flag* |
| **CMC** | *Complement carry flag* |
| **CMP** | *Compare operands* |
| **CMPSB** | *Compare bytes in memory* |
| **CMPSW** | *Compare words* |
| **CWD** | *Convert word to doubleword* |
| **DAA** | *Decimal adjust AL after addition* |
| **DAS** | *Decimal adjust AL after subtraction* |
| **DEC** | *Decrement by 1* |
| **DIV** | *Unsigned divide* |
| **ESC** | *Used with floating-point unit* |

| | |
|---|---|
| **HLT** | *Enter halt state* |
| **IDIV** | *Signed divide* |
| **IMUL** | *Signed multiply* |
| **IN** | *Input from port* |
| **INC** | *Increment by 1* |
| **INT** | *Call to interrupt* |
| **INTO** | *Call to interrupt if overflow* |
| **IRET** | *Return from interrupt* |
| **Jcc** | *Jump if condition* |
| **JMP** | *Jump* |
| **LAHF** | *Load flags into AH register* |
| **LDS** | *Load pointer using DS* |
| **LEA** | *Load Effective Address* |
| **LES** | *Load ES with pointer* |
| **LOCK** | *Assert BUS LOCK# signal* |
| **LODSB** | *Load string byte* |
| **LODSW** | *Load string word* |
| **LOOP/LOOPx** | *Loop control* |
| **MOV** | *Move* |
| **MOVSB** | *Move byte from string to string* |
| **MOVSW** | *Move word from string to string* |
| **MUL** | *Unsigned multiply* |

## Exemple de CPU CISC : Intel 8086

# CISC CPU

# 8086 Instruction set (2/2)

| | |
|---|---|
| **NEG** | *Two's complement negation* |
| **NOP** | *No operation* |
| **NOT** | *Negate the operand, logical NOT* |
| **OR** | *Logical OR* |
| **OUT** | *Output to port* |
| **POP** | *Pop data from stack* |
| **POPF** | *Pop data from flags register* |
| **PUSH** | *Push data onto stack* |
| **PUSHF** | *Push flags onto stack* |
| **RCL** | *Rotate left (with carry)* |
| **RCR** | *Rotate right (with carry)* |
| **REPxx** | *Repeat MOVS/STOS/CMPS/LODS/SCAS* |
| **RET** | *Return from procedure* |
| **RETN** | *Return from near procedure* |
| **RETF** | *Return from far procedure* |
| **ROL** | *Rotate left* |
| **ROR** | *Rotate right* |
| **SAHF** | *Store AH into flags* |
| **SAL** | *Shift Arithmetically left (signed shift left)* |
| **SAR** | *Shift Arithmetically right (signed shift right)* |
| **SBB** | *Subtraction with borrow* |

| | |
|---|---|
| **SCASB** | *Compare byte string* |
| **SCASW** | *Compare word string* |
| **SHL** | *Shift left (unsigned shift left)* |
| **SHR** | *Shift right (unsigned shift right)* |
| **STC** | *Set carry flag* |
| **STD** | *Set direction flag* |
| **STI** | *Set interrupt flag* |
| **STOSB** | *Store byte in string* |
| **STOSW** | *Store word in string* |
| **SUB** | *Subtraction* |
| **TEST** | *Logical compare (AND)* |
| **WAIT** | *Wait until not busy* |
| **XCHG** | *Exchange data* |
| **XLAT** | *Table look-up translation* |
| **XOR** | *Exclusive OR* |

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Prenons un exemple d'instructions CISC 8086.

Les deux codes qui suivent réalisent le même traitement et permettent de déplacer 100 octets en mémoire d'une adresse source vers une adresse destination :

**CISC**

```
MOV     CX, 100
MOV     DI, dst
MOV     SI, src
REP     MOVSB
```

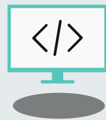
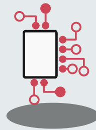
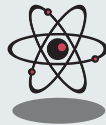
**RISC**

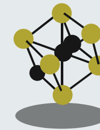
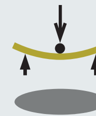
```
        MOV     CX, 100
        MOV     DI, dst
        MOV     SI, src
LOOP:
        MOV     AL, [SI]
        MOV     [DI], AL
        INC     SI
        INC     DI
        DEC     CX
        JNX     LOOP
```

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

# ISA EXTENSIONS

Par abus de langage, les CPU compatibles du jeu d'instructions 80x86 (8086, 80386, 80486, ...) sont nommés CPU **x86**. Depuis l'arrivée d'architectures 64-bit ils sont par abus de langage nommés **x64**.

Pour être rigoureux chez Intel, il faut nommer les jeux d'instructions et CPU 32-bit associés **IA-32** (depuis le 80386 en 1985) et les ISA 64-bit **Intel 64** ou **EM64T** (depuis le Pentium 4 Prescott en 2004).

L'une des grandes forces (et paradoxalement faiblesse) de ce jeu d'instructions est d'assurer une rétrocompatibilité avec les jeux d'instructions d'architectures antérieures.

En contrepartie, il s'agit d'une architecture matérielle très complexe, difficile à accélérer imposant de fortes contraintes de consommation et d'échauffement.

## Jeux d'instructions

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

| CPU Architecture | Nom extension | Instructions |
|---|---|---|
| 8086 Original x86 | - | AAA, AAD, AAM, AAS, ADC, ADD, AND, CALL, CBW, CLC, CLD, CLI, CMC, CMP, CMPSzz, CWD, DAA, DAS, DEC, DIV, ESC, HLT, IDIV, IMUL, IN, INC, INT, INTO, IRET, Jcc, LAHF, LDS, LEA, LES, LOCK, LODSzz, LODSW, LOOPcc, MOV, MOVSzz, MUL, NEG, NOP, NOT, OR, OUT, POP, POPF, PUSH, PUSHF, RCL, RCR, REPcc, RET, RETF, ROL, ROR, SAHF, SAL, SALC, SAR, SBB, SCASzz, SHL, SAL, SHR, STC, STD, STI, STOSzz, SUB, TEST, WAIT, XCHG,XLAT, XOR |
| 80186/80188 | - | BOUND, ENTER, INSB, INSW, LEAVE, OUTSB, OUTSW, POPA, PUSHA, PUSHW |
| 80286 | - | ARPL, CLTS, LAR, LGDT, LIDT, LLDT, LMSW, LOADALL, LSL, LTR, SGDT, SIDT, SLDT, SMSW, STR, VERR, VERW |
| 80386 | - | BSF, BSR, BT, BTC, BTR, BTS, CDQ, CMPSD, CWDE, INSD, IRETD, IRETDF, IRETF, JECXZ, LFS, LGS, LSS, LODSD, LOOPD, LOOPED, LOOPNED, LOOPNZD, LOOPZD, MOVSD, MOVSX, MOVZX, OUTSD, POPAD, POPFD, PUSHAD, PUSHD, PUSHFD, SCASD, SETA, SETAE, SETB, SETBE, SETC, SETE, SETG, SETGE, SETL, SETLE, SETNA, SETNAE, SETNB, SETNBE, SETNC, SETNE, SETNG, SETNGE, SETNL, SETNLE, SETNO, SETNP, SETNS, SETNZ, SETO, SETP, SETPE, SETPO, SETS, SETZ, SHLD, SHRD, STOSD |
| 80486 | - | BSWAP, CMPXCHG, INVD, INVLPG, WBINVD, XADD |
| Pentium | - | **CPUID**, CMPXCHG8B, RDMSR, RDPMC, WRMSR, RSM |
| Pentium pro | - | CMOVA, CMOVAE, CMOVB, CMOVB, CMOVE, CMOVG, CMOVGE, CMOVL, CMOVLE, CMOVNA, CMOVNAE, CMOVNB, CMOVNBE, CMOVNC, CMOVNE, CMOVNG, CMOVNGE, CMOVNL, CMOVNLE, CMOVNO, CMOVNP, CMOVNS, CMOVNZ, CMOVO, CMOVP, CMOVPE, CMOVPO, CMOVS, CMOVZ, RDPMC, SYSENTER, SYSEXIT, UD2 |
| Pentium III | SSE | MASKMOVQ, MOVNTPS, MOVNTQ, PREFETCH0, PREFETCH1, PREFETCH2, PREFETCHNTA, SFENCE |
| Pentium 4 | SSE2 | CLFLUSH, LFENCE, MASKMOVDQU, MFENCE, MOVNTDQ, MOVNTI, MOVNTPD, PAUSE |
| Pentium 4 | SSE3 Hyper Threading | LDDQU, MONITOR, MWAIT |
| Pentium 4 6x2 | VMX | VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON |
| X86-64 | - | CDQE, CQO, CMPSQ, CMPXCHG16B, IRETQ, JRCXZ, LODSQ, MOVSXD, POPFQ, PUSHFQ, **RDTSC**, SCASQ, STOSQ, SWAPGS |
| Pentium 4 | VT-x | VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON |

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Les extensions **x87** ci-dessous n'opèrent que sur des formats flottants.

Historiquement, le 8087 était un coprocesseur externe utilisé comme accélérateur matériel pour des opérations flottantes. Ce coprocesseur fut intégré dans le CPU principal sous forme d'unité d'exécution depuis l'architecture 80486. Cette unité est souvent nommée FPU (*Floating Point Unit*).

| ***CPU Architecture*** | ***Nom extension*** | ***Instructions*** |
|---|---|---|
| ***8087 Original x87*** | - | *F2XM1, FABS, FADD, FADDP, FBLD, FBSTP, FCHS, FCLEX, FCOM, FCOMP, FCOMPP, FDECSTP, FDISI, FDIV, FDIVP, FDIVR, FDIVRP, FENI, FFREE, FIADD, FICOM, FICOMP, FIDIV, FIDIVR, FILD, FIMUL, FINCSTP, FINIT, FIST, FISTP, FISUB, FISUBR, FLD, FLD1, FLDCW, FLDENV, FLDENVW, FLDL2E, FLDL2T, FLDLG2, FLDLN2, FLDPI, FLDZ, FMUL, FMULP, FNCLEX, FNDISI, FNENI, FNINIT, FNOP, FNSAVE, FNSAVEW, FNSTCW, FNSTENV, FNSTENVW, FNSTSW, FPATAN, FPREM, FPTAN, FRNDINT, FRSTOR, FRSTORW, FSAVE, FSAVEW, FSCALE, FSQRT, FST, FSTCW, FSTENV, FSTENVW, FSTP, FSTSW, FSUB, FSUBP, FSUBR, FSUBRP, FTST, FWAIT, FXAM, FXCH, FXTRACT, FYL2X, FYL2XP1* |
| ***80287*** | - | *FSETPM* |
| ***80387*** | - | *FCOS, FLDENVD, FNSAVED, FNSTENVD, FPREM1, FRSTORD, FSAVED, FSIN, FSINCOS, FSTENVD, FUCOM, FUCOMP, FUCOMPP* |
| ***Pentium pro*** | - | *FCMOVB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNBE, FCMOVNE, FCMOVNU, FCMOVU, FCOMI, FCOMIP, FUCOMI, FUCOMIP, FXRSTOR, FXSAVE* |
| ***Pentium 4*** | ***SSE3*** | *FISTTP* |

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Les extensions présentées ci-dessous implémentent toutes des instructions dites SIMD (*Single Instruction Multiple Data*) :

MMX (MultMedia eXtensions) SSE (Streaming SIMD Extensions) AVX (Advanced Vector eXtensions)

| ***CPU Architecture*** | ***Nom extension*** | ***Instructions*** | |
|---|---|---|---|
| ***Pentium MMX*** | ***MMX*** | *EMMS, MOVD, MOVQ, PACKSSDW, PACKSSWB, PACKUSWB, PADDB, PADDD, PADDSB, PADDSW, PADDUSB, PADDUSW, PADDW, PAND, PANDN, PCMPEQB, PCMPEQD, PCMPEQW, PCMPGTB, PCMPGTD, PCMPGTW, PMADDWD, PMULHW, PMULLW, POR, PSLLD, PSLLQ, PSLLW, PSRAD, PSRAW, PSRLD, PSRLQ, PSRLW, PSUBB, PSUBD, PSUBSB, PSUBSW, PSUBUSB, PSUBUSW, PSUBW, PUNPCKHBW, PUNPCKHDQ, PUNPCKHWD, PUNPCKLBW, PUNPCKLDQ, PUNPCKLWD, PXOR* | |
| ***Pentium III*** | ***SSE*** | *Float Inst.* | *ADDPS, ADDSS, CMPPS, CMPSS, COMISS, CVTPI2PS, CVTPS2PI, CVTSI2SS, CVTSS2SI, CVTTPS2PI, CVTTSS2SI, DIVPS, DIVSS, LDMXCSR, MAXPS, MAXSS, MINPS, MINSS, MOVAPS, MOVHLPS, MOVHPS, MOVLHPS, MOVLPS, MOVMSKPS, MOVNTPS, MOVSS, MOVUPS, MULPS, MULSS, RCPPS, RCPSS, RSQRTPS, RSQRTSS, SHUFPS, SQRTPS, SQRTSS, STMXCSR, SUBPS, SUBSS, UCOMISS, UNPCKHPS, UNPCKLPS* |
| | | *Integer Inst.* | *ANDNPS, ANDPS, ORPS, PAVGB, PAVGW, PEXTRW, PINSRW, PMAXSW, PMAXUB, PMINSW, PMINUB, PMOVMSKB, PMULHUW, PSADBW, PSHUFW, XORPS* |
| ***Pentium 4*** | ***SSE2*** | *Float Inst.* | *ADDPD, ADDSD, ANDNPD, ANDPD, CMPPD, CMPSD, COMISD, CVTDQ2PD, CVTDQ2PS, CVTPD2DQ, CVTPD2PI, CVTPD2PS, CVTPI2PD, CVTPS2DQ, CVTPS2PD, CVTSD2SI, CVTSD2SS, CVTSI2SD, CVTSS2SD, CVTTPD2DQ, CVTTPD2PI, CVTTPS2DQ, CVTTSD2SI, DIVPD, DIVSD, MAXPD, MAXSD, MINPD, MINSD, MOVAPD, MOVHPD, MOVLPD, MOVMSKPD, MOVSD, MOVUPD, MULPD, MULSD, ORPD, SHUFPD, SQRTPD, SQRTSD, SUBPD, SUBSD, UCOMISD, UNPCKHPD, UNPCKLPD, XORPD* |
| | | *Integer Inst.* | *MOVDQ2Q, MOVDQA, MOVDQU, MOVQ2DQ, PADDQ, PSUBQ, PMULUDQ, PSHUFHW, PSHUFLW, PSHUFD, PSLLDQ, PSRLDQ, PUNPCKHQDQ, PUNPCKLQDQ* |
| | ***SSE3*** | *Float Inst.* | *ADDSUBPD, ADDSUBPS, HADDPD, HADDPS, HSUBPD, HSUBPS, MOVDDUP, MOVSHDUP, MOVSLDUP* |

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Les instructions et opérandes usuellement manipulées par grand nombre de CPU sur le marché sont dites **scalaires**. Nous parlerons de processeur scalaire (PIC de Microchip, 8051 de Intel, AVR de Atmel, C5xxx de TI…). Par exemple sur 8086 de Intel, prenons l'exemple d'une addition : scalaire + scalaire = scalaire :

```
add     %bl,%al
```

A titre indicatif, les instructions MMX, SSE, AVX, AES … sont dîtes **vectorielles**. Les opérandes ne sont plus des grandeurs scalaires mais des grandeurs vectorielles. Nous parlerons de processeur vectoriel (d'autres architectures vectorielles existent). Prenons un exemple d'instruction vectorielle SIMD SSE4.1, vecteur · vecteur = scalaire :

```
dpps     0xF1,%xmm2,%xmm1
```

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Cette instruction vectorielle peut notamment être très intéressante pour des applications de traitement numérique du signal :

dpps signifie *Dot Product of Packed Single-precision-floating-point-values*, soit produit scalaire sur un paquet de données au format flottant en simple précision (IEEE-754).

Observons le descriptif de l'instruction ainsi qu'un exemple :

**DPPS — Dot Product of Packed Single Precision Floating-Point Values**

| Opcode/ Instruction | Op/ En | 64/32-bit Mode | CPUID Feature Flag | Description |
|---|---|---|---|---|
| 66 0F 3A 40 /r ib<br>DPPS *xmm1, xmm2/m128, imm8* | RMI | V/V | SSE4_1 | Selectively multiply packed SP floating-point values from *xmm1* with packed SP floating-point values from *xmm2*, add and selectively store the packed SP floating-point values or zero values to *xmm1*. |

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

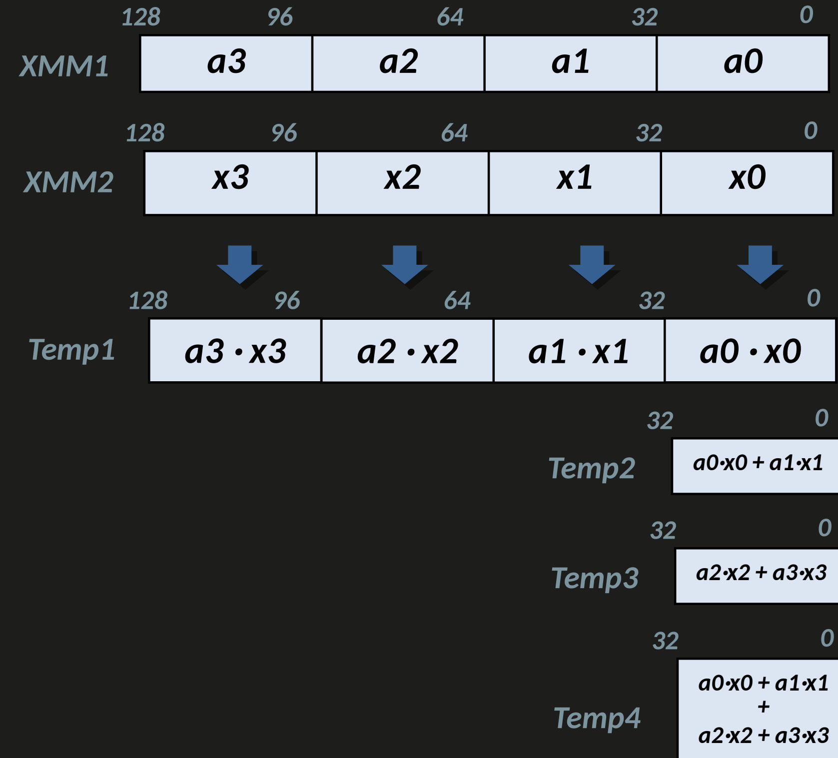
Étudions un exemple d'exécution de l'instruction dpps :

```
dpps    0xF1,%xmm2,%xmm1
```

**Operation**

```
DP_primitive (SRC1, SRC2)
IF (imm8[4] = 1)
    THEN Temp1[31:0] ← DEST[31:0] * SRC[31:0];
    ELSE Temp1[31:0] ← +0.0; FI;
IF (imm8[5] = 1)
    THEN Temp1[63:32] ← DEST[63:32] * SRC[63:32];
    ELSE Temp1[63:32] ← +0.0; FI;
IF (imm8[6] = 1)
    THEN Temp1[95:64] ← DEST[95:64] * SRC[95:64];
    ELSE Temp1[95:64] ← +0.0; FI;
IF (imm8[7] = 1)
    THEN Temp1[127:96] ← DEST[127:96] * SRC[127:96];
    ELSE Temp1[127:96] ← +0.0; FI;

Temp2[31:0] ← Temp1[31:0] + Temp1[63:32];
Temp3[31:0] ← Temp1[95:64] + Temp1[127:96];
Temp4[31:0] ← Temp2[31:0] + Temp3[31:0];
```



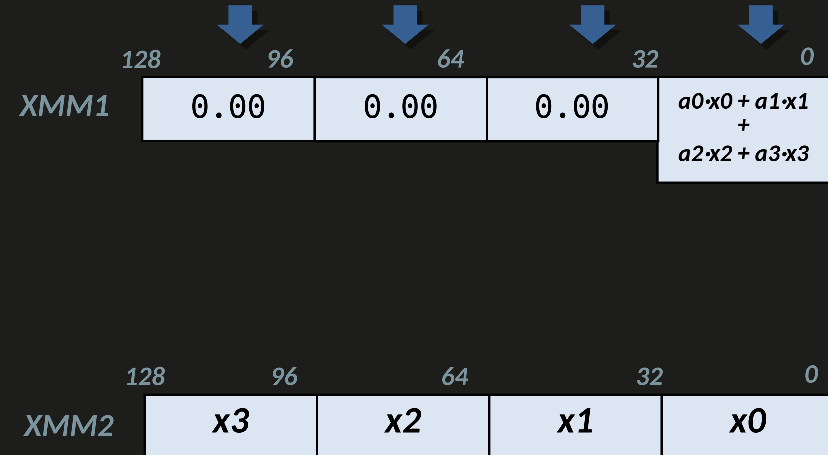
Étudions un exemple d'exécution de l'instruction dpps :

```
dpps      0xF1,%xmm2,%xmm1
```

```
IF (imm8[0] = 1)
    THEN DEST[31:0] ← Temp4[31:0];
    ELSE DEST[31:0] ← +0.0; FI;
IF (imm8[1] = 1)
    THEN DEST[63:32] ← Temp4[31:0];
    ELSE DEST[63:32] ← +0.0; FI;
IF (imm8[2] = 1)
    THEN DEST[95:64] ← Temp4[31:0];
    ELSE DEST[95:64] ← +0.0; FI;
IF (imm8[3] = 1)
    THEN DEST[127:96] ← Temp4[31:0];
    ELSE DEST[127:96] ← +0.0; FI;
```

**DPPS (128-bit Legacy SSE version)**
DEST[127:0]←DP_Primitive(SRC1[127:0], SRC2[127:0]);
DEST[VLMAX-1:128] (Unmodified)

*XMMi (i = 0 to 15 with Intel 64)*
*128-bit General Purpose Registers*
*for SIMD Execution Units*



ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Les extensions x86-64 présentées jusqu'à maintenant ne présentent que les évolutions des jeux d'instructions apportées par Intel.

Les extensions amenées par AMD ne seront pas présentées (MMX+, K6-2, 3DNow, 3DNow!+, SSE4a, …).

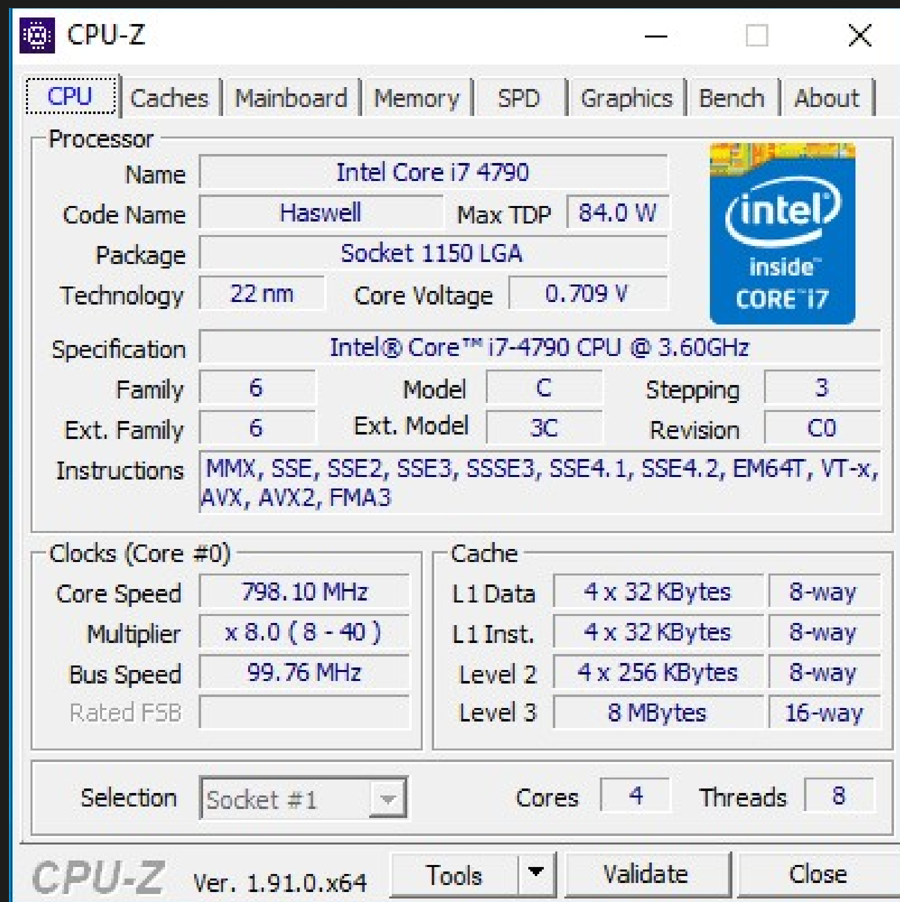
| *CPU Architecture* | *Nom extension* | *Instructions* |
|---|---|---|
| *Core2* | *SSSE3* | *PSIGNW, PSIGND, PSIGNB, PSHUFB, PMULHRSW, PMADDUBSW, PHSUBW, PHSUBSW, PHSUBD, PHADDW, PHADDSW, PHADDD, PALIGNR, PABSW, PABSD, PABSB* |
| *Core2 (45nm)* | *SSE4.1* | *MPSADBW, PHMINPOSUW, PMULLD, PMULDQ, **DPPS**, DPPD, BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, PBLENDVB, PBLENDW, PMINSB, PMAXSB, PMINUW, PMAXUW, PMINUD, PMAXUD, PMINSD, PMAXSD, ROUNDPS, ROUNDSS, ROUNDPD, ROUNDSD, INSERTPS, PINSRB, PINSRD/PINSRQ, EXTRACTPS, PEXTRB, PEXTRW, PEXTRD/PEXTRQ, PMOVSXBW, PMOVZXBW, PMOVSXBD, PMOVZXBD, PMOVSXBQ, PMOVZXBQ, PMOVSXWD, PMOVZXWD, PMOVSXWQ, PMOVZXWQ, PMOVSXDQ, PMOVZXDQ, PTEST, PCMPEQQ, PACKUSDW, MOVNTDQA* |
| *Nehalem* | *SSE4.2* | *CRC32, PCMPESTRI, PCMPESTRM, PCMPISTRI, PCMPISTRM, PCMPGTQ* |
| *Sandy Bridge* | *AVX* | *VFMADDPD, VFMADDPS, VFMADDSD, VFMADDSS, VFMADDSUBPD, VFMADDSUBPS, VFMSUBADDPD, VFMSUBADDPS, VFMSUBPD, VFMSUBPS, VFMSUBSD, VFMSUBSS, VFNMADDPD, VFNMADDPS, VFNMADDSD, VFNMADDSS, VFNMSUBPD, VFNMSUBPS, VFNMSUBSD, VFNMSUBSS* |
| *Nehalem* | *AES* | *AESENC, AESENCLAST, AESDEC, AESDECLAST, AESKEYGENASSIST, AESIMC* |

## Infos sur le processeur : instruction CPUID

L'instruction **CPUID** arrivée avec l'architecture Pentium permet de récupérer très facilement toutes les informations relatives à l'architecture matérielle du GPP (CPUs, caches, adressage virtuel …).

L'utilitaire libre CPU-Z utilise notamment ce registre pour retourner des informations sur l'architecture :

https://www.cpuid.com/softwares/cpu-z.html



## Infos sur le processeur : lscpu

Sous Linux, la commande `lscpu` permet d'afficher les informations relatives au processeur équipant l'ordinateur.

`lscpu` = « `ls` » + « cpu »

dboudier@dboudier-Precision-3541: ~

```
dboudier:~$ lscpu
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          16
On-line CPU(s) list:             0-15
Thread(s) per core:              2
Core(s) per socket:              8
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           158
Model name:                      Intel(R) Core(TM) i9-9880H CPU @ 2.3
                                 0GHz
Stepping:                        13
CPU MHz:                         2300.000
CPU max MHz:                     4800,0000
CPU min MHz:                     800,0000
BogoMIPS:                        4599.93
Virtualization:                  VT-x
L1d cache:                       256 KiB
L1i cache:                       256 KiB
L2 cache:                        2 MiB
L3 cache:                        16 MiB
NUMA node0 CPU(s):               0-15
Vulnerability Gather data sampling: Mitigation; Microcode
Vulnerability Itlb multihit:     KVM: Mitigation: VMX disabled
Vulnerability L1tf:              Not affected
Vulnerability Mds:               Not affected
Vulnerability Meltdown:          Not affected
Vulnerability Mmio stale data:   Mitigation; Clear CPU buffers; SMT v
                                 ulnerable
Vulnerability Retbleed:          Mitigation; Enhanced IBRS
Vulnerability Spec store bypass: Mitigation; Speculative Store Bypass
                                  disabled via prctl and seccomp
Vulnerability Spectre v1:        Mitigation; usercopy/swapgs barriers
                                  and __user pointer sanitization
Vulnerability Spectre v2:        Mitigation; Enhanced IBRS, IBPB cond
```

## Infos sur le processeur : cpuinfo

Mais la méthode la plus connue consiste à afficher le contenu du fichier cpuinfo :

`cat /proc/cpuinfo`

Cette méthode fournit bien plus d'informations puisqu'elle liste chaque cœur logique du GPP.

```
dboudier@dboudier-Precision-3541: ~
dboudier:~$ cat /proc/cpuinfo | more
processor       : 0
vendor_id       : GenuineIntel
cpu family      : 6
model           : 158
model name      : Intel(R) Core(TM) i9-9880H CPU @ 2.30GHz
stepping        : 13
microcode       : 0xfa
cpu MHz         : 2300.000
cache size      : 16384 KB
physical id     : 0
siblings        : 16
core id         : 0
cpu cores       : 8
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 22
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge m
ca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscal
l nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good n
opl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor
ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2
x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm
 abm 3dnowprefetch cpuid_fault epb invpcid_single ssbd ibrs ibpb stibp i
brs_enhanced tpr_shadow vnmi flexpriority ept vpid ept_ad fsgsbase tsc_a
djust sgx bmi1 avx2 smep bmi2 erms invpcid mpx rdseed adx smap clflushop
t intel_pt xsaveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hw
p_notify hwp_act_window hwp_epp sgx_lc md_clear flush_l1d arch_capabilit
ies
vmx flags       : vnmi preemption_timer invvpid ept_x_only ept_ad ept_1g
b flexpriority tsc_offset vtpr mtf vapic ept vpid unrestricted_guest ple
 shadow_vmcs pml ept_mode_based_exec
bugs            : spectre_v1 spectre_v2 spec_store_bypass swapgs taa itl
b_multihit srbds mmio_stale_data retbleed eibrs_pbrsb gds
bogomips        : 4599.93
clflush size    : 64
```

## Manuel du développeur

De même, lorsque l'on est amené à développer sur un processeur donné, il est essentiel de travailler avec les documents de référence proposés par le fondeur, Intel dans notre cas. Vous pouvez télécharger les différents documents de référence à cette URL :

http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html

Seulement 5066 pages dans sa version 2023

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Combined Volumes:
1, 2A, 2B, 2C, 2D, 3A, 3B, 3C, 3D, and 4

**NOTE:** This document contains all four volumes of the Intel 64 and IA-32 Architectures Software Developer's Manual: *Basic Architecture*, Order Number 253665; *Instruction Set Reference A-Z*, Order Number 325383; *System Programming Guide*, Order Number 325384; *Model-Specific Registers*, Order Number 335592. Refer to all four volumes when evaluating your design needs.

# CONTACT

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

Dimitri Boudier – PRAG ENSICAEN

dimitri.boudier@ensicaen.fr

Avec l'aide précieuse de :

- Hugo Descoubes (PRAG ENSICAEN)

Except where otherwise noted, this work is licensed under
https://creativecommons.org/licenses/by-nc-sa/4.0/