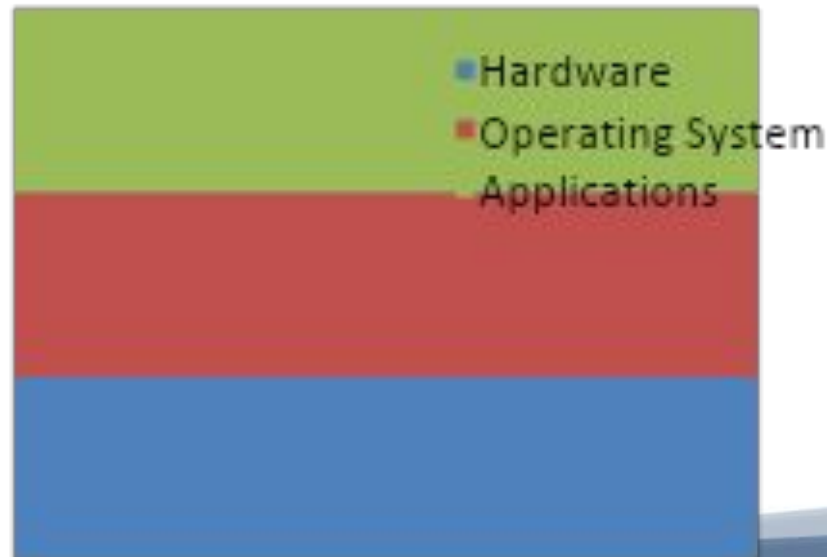


# ***LANGAGE D'ASSEMBLAGE***

---

***Architecture et Technologie des Ordinateurs***

*Un système travaillant sur une architecture à CPU peut très souvent être découpé en couche. Il s'agit d'une solution mixte logicielle (OS et applications) et matérielle. Un développeur logiciel dit "bas niveau" travaille dans les couches basses de ce modèle, typiquement au niveau du système d'exploitation.*



*Observons en quelques chiffres, la répartition des marchés des systèmes d'exploitation sur quelques un des principaux grands domaines d'application :*

- **Windows de Microsoft** : ~90% du marché des ordinateurs personnels en 2008, ~20% des appareils personnels en 2013, 38% des serveurs en 2012



- **UNIX (GNU/Linux, iOS, MAC OS X, Android ...)**: 90% du marché des Smartphones en 2012 (Android ~75%), 31% des serveurs en 2012, GNU/Linux ~95% des superordinateurs



iOS





***Vous aurez un enseignement dédié aux systèmes d'exploitation en 2A. Cet enseignement sera assuré par M. Sébastien Fourey.***

*Malheureusement, développement bas niveau ne veut pas dire développement simple. Un ingénieur travaillant dans ce domaine doit notamment être compétent sur les points suivants :*

- ***Architectures matérielles*** (CPU, hiérarchie et gestion mémoire, gestion périphériques, mécanismes d'optimisations ...)
- ***Langages de programmation*** (essentiellement C/C++ et ASM ou assembleur)
- ***Outils de Développement Logiciel*** (IDE, chaîne de compilation C, outils de debuggage et de profilage, programmation système d'exploitation ...)

Effectuons quelques rappels sur une chaîne de compilation C (C toolChain ou C toolSuite). **Les slides qui suivent sont à savoir par cœur.**

Les exemples suivants sont donnés sous la chaîne de compilation GCC (GNU Compilation Collection). L'architecture est la même que toute autre toolChain C, cependant les extensions des fichiers ne sont pas standardisées et peuvent changer d'une chaîne à une autre ou d'une plateforme matérielle à une autre.

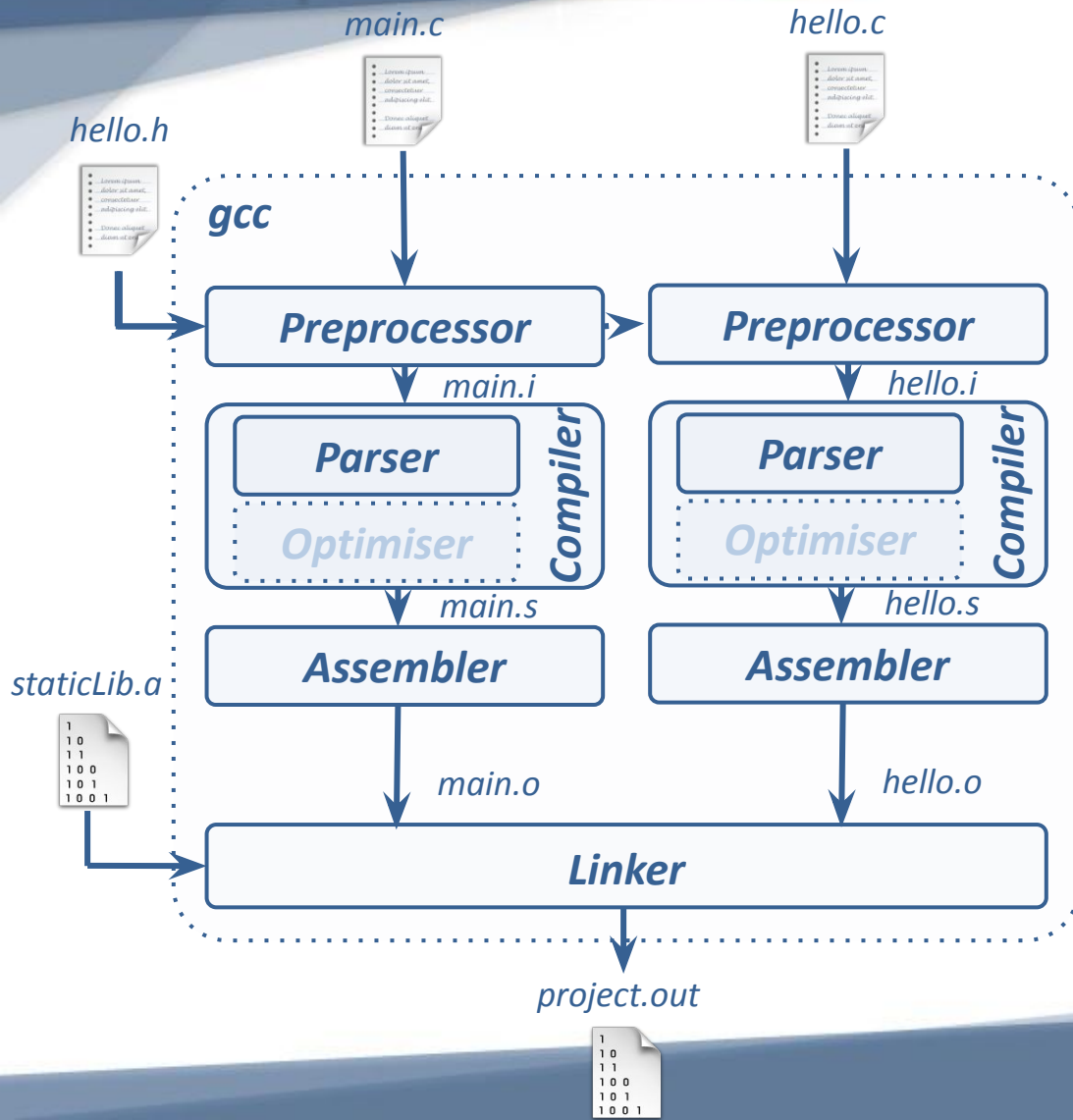


<http://gcc.gnu.org/>



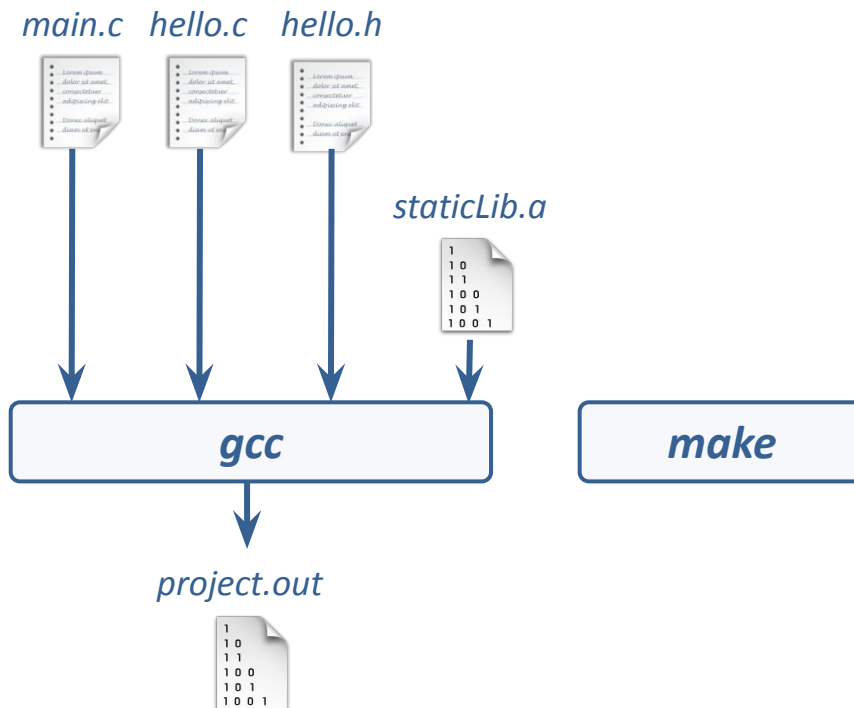


***Cet enseignement s'appuie sur les compétences enseignées dans les enseignements "Outils de Développement Logiciel" et "Programmation et langage C" respectivement assurés par M. Jalal Fadili et M. Emmanuel Sibille.***

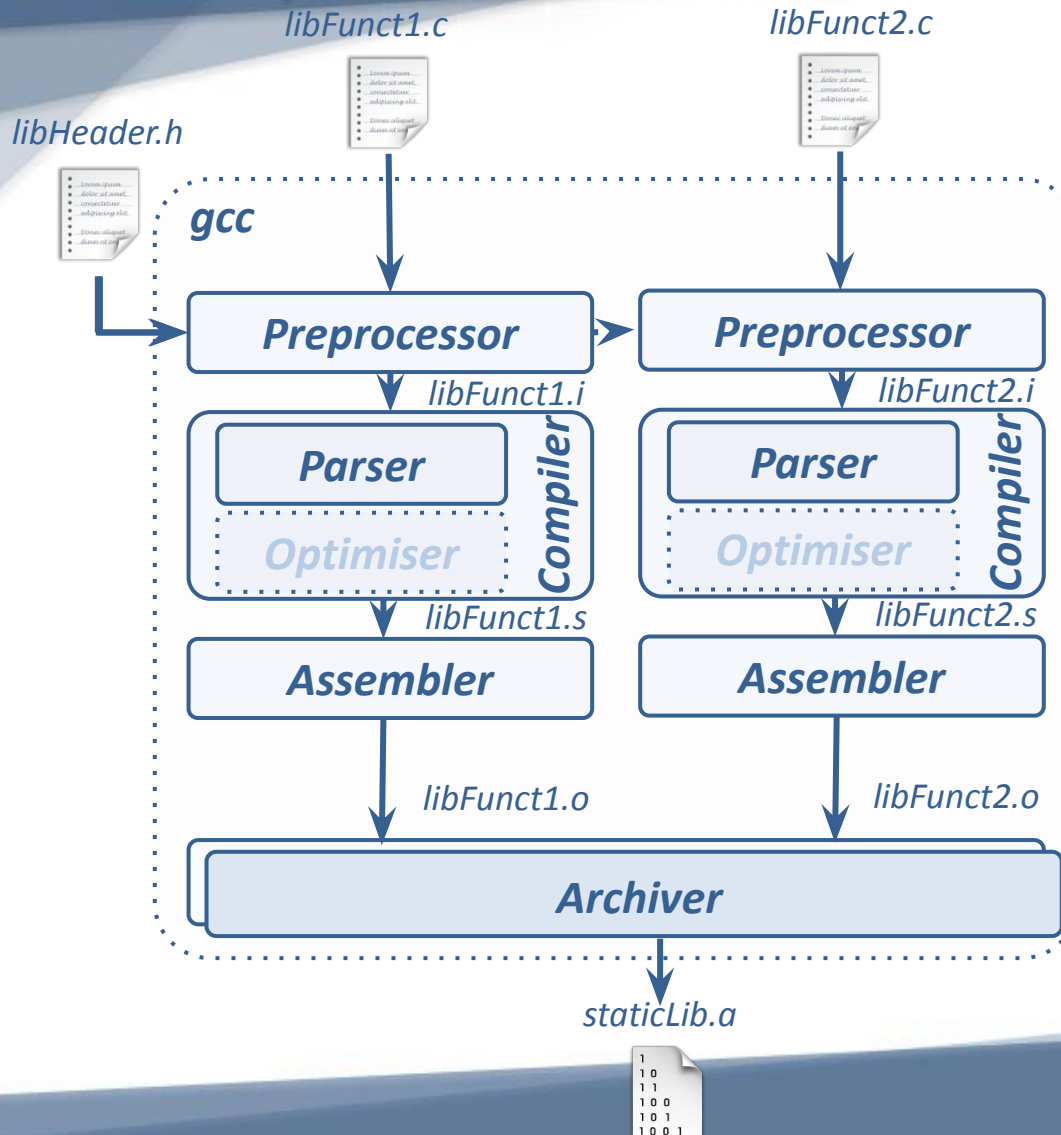


- **Préprocesseur** : interprétation directives compilation (#) et suppression commentaires
- **Compilateur** : Analyse programme (lexicale, syntaxique, sémantique..), code intermédiaire, optimisation optionnelle, génération ASM (CPU architecture dépendant).
- **Assembleur** : Génération code binaire/objet relogeable pour CPU cible
- **Editeur de liens** : Liens entre fichiers objets et bibliothèques (statique et/ou dynamique). Génération et chargement code binaire/exécutable absolu

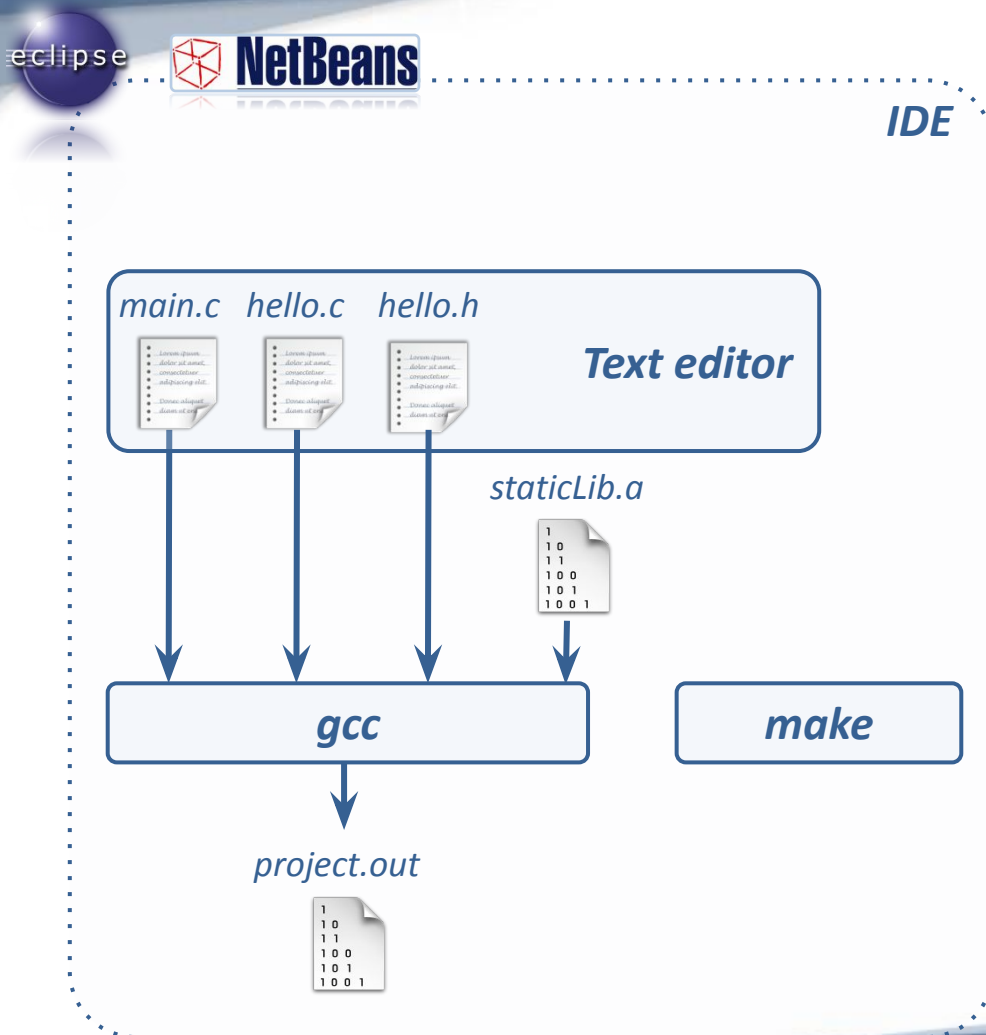




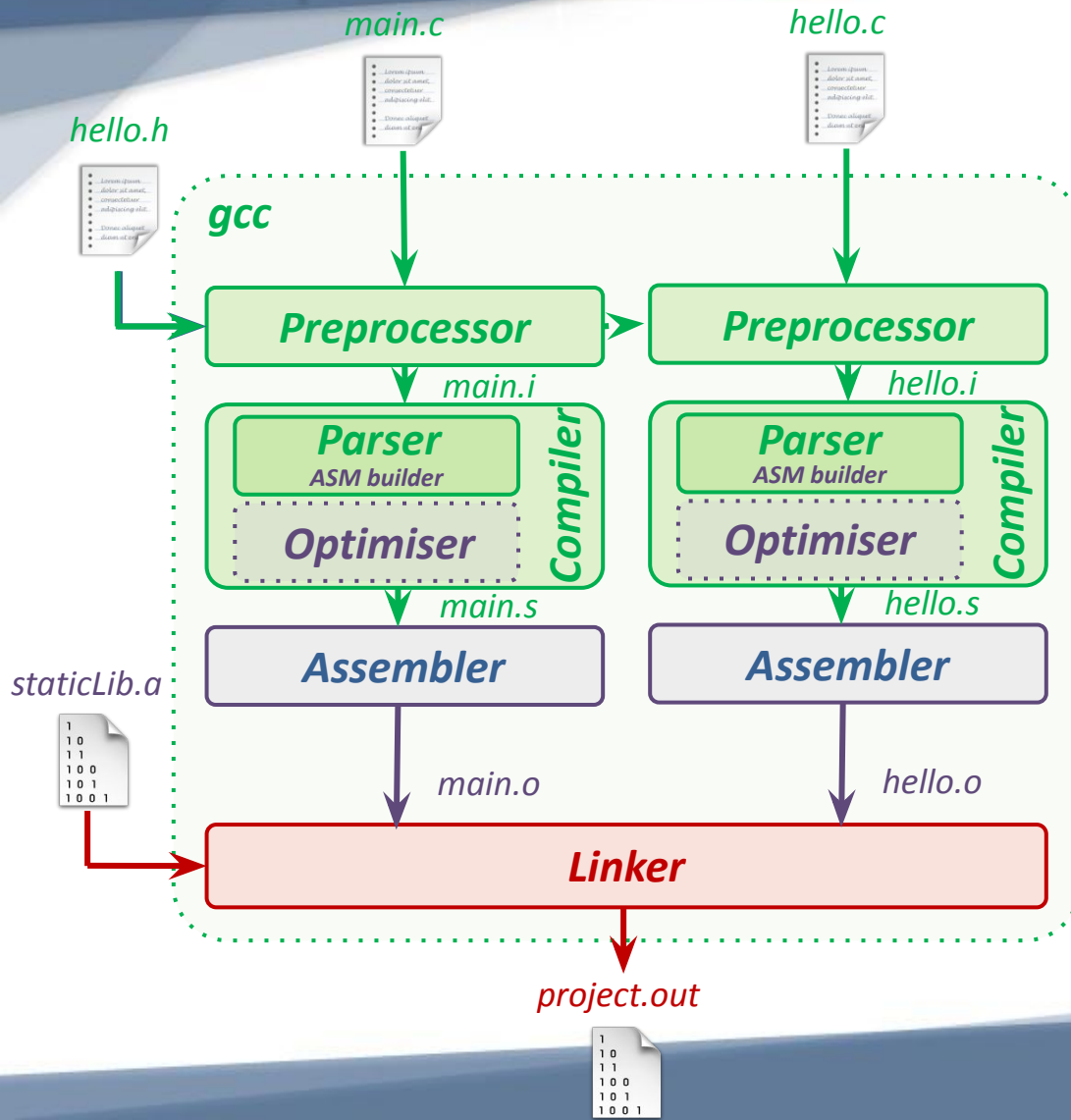
- ***make*** : utilitaire de programmation pour l'automatisation de procédures de compilation
- ***Archiver*** : construction de bibliothèques statiques. Archive réalisée à partir de fichiers objets



- **make** : utilitaire de programmation pour l'automatisation de procédures de compilation
- **Archiver** : construction de bibliothèques statiques. Archive réalisée à partir de fichiers objets



- **make** : utilitaire de programmation pour l'automatisation de procédures de compilation
- **Archiver** : construction de bibliothèques statiques. Archive réalisée à partir de fichiers objets
- **Integrated Development Environment** : Aide au développement logiciel. Intègre généralement un éditeur de texte, automatisation procédure de construction de code, debugger...



- Les 2 premières étapes de la compilation sont “en partie” architecture agnostique.



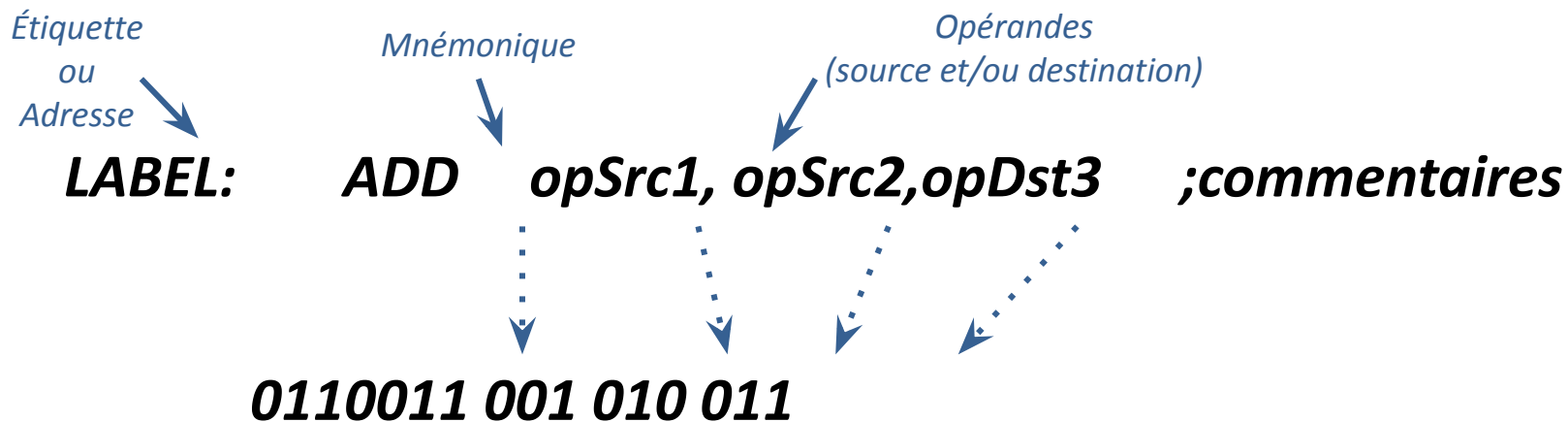
***Vous aurez un enseignement dédié à la compilation en 2A (compilation et théorie des langages). Cet enseignement sera assuré par M. Régis Clouard.***

- Les étapes de compilation suivantes dépendent de l'architecture CPU (optimiser et assembler) et du mapping mémoire (linker). Nous nous intéresserons donc tout particulièrement à ces étapes

*Observons les 3 trois principaux environnements de compilation utilisés sur architecture x86 :*

- ***Visual Studio*** proposé par Windows
- ***Intel C++ Compiler XE*** proposé par Intel
- ***GCC (GNU Compiler Collection)*** issu du monde de l'Open Source ayant vocation à être multiplateforme (cross-compilation ARM, MIPS ...). Les deux principaux environnement de compilation rencontrés sous Windows sont Cygwin et MinGW.

**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 ou code machine. Prenons l'exemple d'une instruction assembleur élémentaire raccrochée à aucune architecture connue :**





*Hormis label et commentaires, en général à tout champ d'une instruction assembleur correspond un champ dans le code binaire équivalent. Ce code binaire ne peut être compris et interprété que par le CPU cible.*

***LABEL:    ADD    opSrc1, opSrc2, opDst3    ;commentaires***

***0110011 001 010 011***

*N'est utilisé que par  
les instructions de  
saut*

*Opcode  
ou  
Code opératoire*

*L'assembleur est probablement le langage de programmation le moins universel au monde. Il existe autant de langage d'assemblage que de familles de CPU. Prenons l'exemple des jeux d'instructions Cortex-Mx de ARM. La société Anglaise ARM propose à elle seule 3 familles de CPU, cortex-M, -R, -A possédant chacune des sous familles. Ne regardons que la sous famille cortex-M :*

Cortex-Mx ARM Instruction set

Cortex-M0/M1																Cortex-M3																Cortex-M4															
<b>ARM</b>																<b>ARM</b>																<b>ARM</b>															
<b>YABS</b> <b>YADD</b> <b>YCHP</b> <b>YCHPE</b> <b>YCVT</b> <b>YCVTR</b> <b>YDIV</b> <b>YLDH</b> <b>VLDL</b> <b>VHLA</b> <b>VHLS</b> <b>VHOV</b> <b>VHRS</b> <b>VHSL</b> <b>VNEG</b> <b>VNMLA</b> <b>VNMLS</b> <b>VNMUL</b> <b>VPOP</b> <b>VPUSH</b> <b>VQRT</b> <b>VSTM</b> <b>VSTR</b> <b>VSUB</b> <b>VFHA</b> <b>VFMS</b> <b>VFNHA</b> <b>VFNMS</b>																<b>PKH</b> <b>QADD</b> <b>QADDH</b> <b>QADDL</b> <b>QASX</b> <b>QDADD</b> <b>QDSUB</b> <b>QSAX</b> <b>QSUB</b> <b>QSUBH</b> <b>QSUBL</b> <b>SADDH</b> <b>SASX</b> <b>SADDD</b> <b>SEL</b> <b>SHADDH</b> <b>SHADDE</b> <b>SHASX</b> <b>SHSAX</b> <b>SHSUBH</b> <b>SHSUBL</b> <b>SHLASH</b> <b>SHLBT</b> <b>SHLATB</b> <b>SHLATL</b> <b>SHLAD</b> <b>SHLALB</b> <b>SHLALB</b> <b>SHLALB</b> <b>SHLALT</b> <b>SHLALD</b> <b>SHLAWB</b> <b>SHLAWT</b> <b>SHLD</b> <b>SHSLD</b> <b>SHMLA</b> <b>SHMLS</b> <b>SHMUL</b> <b>SHUAD</b> <b>SHULB</b>																<b>ADC</b> <b>ADD</b> <b>ADR</b> <b>AND</b> <b>ASR</b> <b>B</b> <b>SHLBT</b> <b>SHULT</b> <b>CLZ</b> <b>BFC</b> <b>BFI</b> <b>BIC</b> <b>CBP</b> <b>CLREX</b> <b>SHULTB</b> <b>SHULWT</b> <b>CBNZ</b> <b>CBZ</b> <b>CHN</b> <b>CHP</b> <b>DBG</b> <b>EOR</b> <b>SHUJWB</b> <b>SHUSD</b> <b>LDHIA</b> <b>LDHOB</b> <b>LDR</b> <b>LDHB</b> <b>LDHBT</b> <b>LDHD</b> <b>SSATB</b> <b>SSAX</b> <b>LDREX</b> <b>LDREXB</b> <b>LDREXH</b> <b>LDRH</b> <b>LDRHT</b> <b>LDRSB</b> <b>SSUBH</b> <b>SSUBL</b> <b>LDRSBT</b> <b>LDRSHT</b> <b>LDRSH</b> <b>LDRT</b> <b>MCR</b> <b>LSL</b> <b>SXTAB</b> <b>SXTABH</b> <b>LSR</b> <b>MCR</b> <b>MLS</b> <b>MLA</b> <b>MOV</b> <b>MOVT</b> <b>SXTAB</b> <b>SXTABH</b> <b>MRC</b> <b>MRC</b> <b>HUL</b> <b>MLA</b> <b>MOV</b> <b>DRN</b> <b>SXTAB</b> <b>SXTABH</b> <b>ORR</b> <b>PLD</b> <b>PLDW</b> <b>PLI</b> <b>POP</b> <b>PUSH</b> <b>UADDB</b> <b>UADDB</b> <b>RSIT</b> <b>REV</b> <b>REVH</b> <b>REVSH</b> <b>ROR</b> <b>RRX</b> <b>UASX</b> <b>UHADDH</b> <b>RSB</b> <b>SBC</b> <b>SBC</b> <b>SBC</b> <b>SBC</b> <b>SBC</b> <b>UHADDL</b> <b>UHADDL</b> <b>SDIV</b> <b>SEV</b> <b>SHLAL</b> <b>SHLAL</b> <b>SHLAL</b> <b>SHLAL</b> <b>UHSAX</b> <b>UHSUBH</b> <b>SHULL</b> <b>SSAT</b> <b>STC</b> <b>STR</b> <b>STR</b> <b>STR</b> <b>UHSUBL</b> <b>UMAAL</b> <b>STHIA</b> <b>STHOB</b> <b>STR</b> <b>STR</b> <b>STR</b> <b>STR</b> <b>UQADDB</b> <b>UQADDB</b> <b>STRB</b> <b>STRBT</b> <b>STRD</b> <b>STRD</b> <b>STRD</b> <b>STRD</b> <b>UQASX</b> <b>UQASX</b> <b>STREX</b> <b>STREXB</b> <b>STREXH</b> <b>STREX</b> <b>STREX</b> <b>STREX</b> <b>UQSUBH</b> <b>UQSUBH</b> <b>STRH</b> <b>STRHT</b> <b>STRT</b> <b>STRT</b> <b>STRT</b> <b>STRT</b> <b>USADB</b> <b>USADB</b> <b>SUB</b> <b>SXTB</b> <b>SXTB</b> <b>SXTB</b> <b>SXTB</b> <b>SXTB</b> <b>USATB</b> <b>USATB</b> <b>TBB</b> <b>TBB</b> <b>TEQ</b> <b>TEQ</b> <b>TEQ</b> <b>TEQ</b> <b>USUBH</b> <b>USUBH</b> <b>TST</b> <b>UBFX</b> <b>UDIV</b> <b>UDIV</b> <b>UDIV</b> <b>UDIV</b> <b>UXTAB</b> <b>UXTABH</b> <b>UHAL</b> <b>UHAL</b> <b>USAT</b> <b>USAT</b> <b>USAT</b> <b>USAT</b> <b>UXTAB</b> <b>UXTABH</b> <b>UXTB</b> <b>UXTB</b> <b>WFE</b> <b>WFE</b> <b>WFE</b> <b>WFE</b> <b>UXTAB</b> <b>UXTABH</b> <b>WFI</b> <b>YIELD</b> <b>IT</b> <b>IT</b> <b>IT</b> <b>IT</b> <b>UXTAB</b> <b>UXTABH</b>															

*Observons les principaux acteurs dans le domaine des CPU's. Chaque fondeur présenté ci-dessous propose une voire plusieurs architectures de CPU qui lui sont propres et possédant donc les jeux d'instructions associés (CPU server et mainframe non présentés) :*

- ***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) ...

***Tout CPU est capable de décoder puis d'exécuter un jeu d'instruction qui lui est propre (ou instruction set ou ISA ou Instruction Set Architecture). Dans tous les cas, ces instructions peuvent être classées en grandes 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 (16bits et plus) voire flottants peuvent l'être également.
- ***Management de données*** : déplacement de données dans l'architecture matérielle (CPU vers CPU, CPU vers mémoire ou mémoire vers 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. Nous pouvons rendre ces sauts conditionnels à l'aide d'opérations arithmétiques et logiques ou de comparaisons.

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...) ...



- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

**Les jeux d'instructions peuvent être classés en 2 grandes familles, RISC et CISC**, respectivement Reduce et Complex Instruction Set Computer. Les architectures RISC n'implémentent en général que des instructions élémentaires (CPU's ARM, MIPS, 8051, PIC18 ...). A l'inverse, les architectures CISC (CPU's 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 DSP RISC-like TMS320C66xx de TI compte 323 instructions. Néanmoins, les architectures compatibles x86-64 sont des architectures CISC. Nous allons rapidement comprendre pourquoi.



- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

### **Avantages architecture CISC :**

- Empreinte mémoire programme faible, donc plus d'instructions contenues en cache. Néanmoins sur CPU CISC, plus 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)

- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

### **Inconvénients architecture RISC :**

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

### **Avantages architecture RISC :**

- *Architecture CPU moins complexe (mécanismes d'accélération matériels, décodeurs, Execution Units ...).*
- *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.*

- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

*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 MCU's ou ASIC's (licence libre). Prenons quelques exemples de fondeurs les utilisant : NXP, silabs, Atmel ...*

**8051 Intel CPU (only CPU)**  
(1980)



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



- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

8051 Instruction set

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

- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

*Observons le jeu d'instructions complet d'un CPU 16bits CISC 8086 proposé par Intel en 1978. Il s'agit du premier processeur de la famille x86. En 2012, un core i7 est toujours capable d'exécuter le jeu d'instruction d'un 8086. Bien sûr, la réciproque n'est pas vraie.*

**8086 Intel CPU**  
(1978)





- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

Original 8086 Instruction set

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

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



- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

Original 8086 Instruction set

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

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

- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

Prenons un exemple d'instruction 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
BE6 10A2B
```

### 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
```

```
MOV CX,100
DEC CX
INC DI
```

- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

*Attention, si vous lisez de l'assembleur x86-64, il existe deux syntaxes très répandues. La syntaxe Intel et la syntaxe AT&T utilisée par défaut par gcc (systèmes UNIX).*

### Intel Syntax

```
MOV ebx,0FAh
```

### AT&T Syntax

```
MOV $0xFA, %ebx
```

#### Syntaxe AT&T :

- Opérandes sources à gauche et destination à droite
- Constantes préfixées par \$ (adressage immédiat)
- Constantes écrites avec syntaxe langage C (0x + valeur = hexadécimal)
- Registres préfixés par %
- Segmentation : [ds:20] devient %ds:20, [ss:bp] devient %ss:%bp ...

- Adressage indirect [ebx] devient (%ebx), [ebx + 20h] devient 0x20(%ebx), [ebx+ecx\*2h-1Fh] devient -0x1F(%ebx, %ecx, 0x2) ...
- Suffixes, b=byte=1o, w=word=2o, s=short=4o, l=long=4o, q=quad=8o, t=ten=10o, o=octo=16o=128bits (x64)
- ...

- Jeu d'instruction RISC 8051
- Jeu d'instruction CISC 8086

Prenons un exemple de code écrit dans les 2 syntaxes :

### Intel Syntax

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

```
JNX     700b
DEC     CX
INC     DI
```

### AT&T Syntax

```
movw    $100, %cx
movw    dst, %di
movw    src, %di
LOOP:
movb    (%si), %al
movb    %al, (%di)
inc     %si
inc     %di
dec     %cx
jnx     LOOP
```

```
jux     700b
qec     %cx
inc     %di
```

*Par abus de langage, les CPU compatibles du jeu d'instruction 80x86 (8086, 80386, 80486..) sont nommés CPU x86. Depuis l'arrivée d'architectures 64bits ils sont par abus de langage nommés x64. Pour être rigoureux chez Intel, il faut nommer les jeux d'instructions et CPU 32bits associés IA-32 (depuis le 80386 en 1985) et les ISA 64bits Intel 64 ou EM64T (depuis le Pentium 4 Prescott en 2004).*



*L'une des grandes forces (et paradoxalement faiblesse) de ce jeu d'instruction 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.*



## Extensions x86 et x64 n'opérant que sur des formats entiers :

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, IRETDI, IRETF, JECXZ, LFS, LGS, LSS, LODSD, LOOPD, LOOPED, LOOPNE, LOOPNZ, LOOPZD, MOVSD, MOVSB, MOVSDI, MOVZX, OUTSD, POPAD, POPFD, PUSHAD, PUSHDI, 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, CMOVQ, CMOVPE, CMOVPO, CMOVQ, 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, MOVSD, POPFQ, PUSHFQ, RDTSC, SCASQ, STOSQ, SWAPGS
Pentium 4	VT-x	VMPTRLD, VMPTRST, VMCLEAR, VMREAD, VMWRITE, VMCALL, VMLAUNCH, VMRESUME, VMXOFF, VMXON



*Les extensions x87 ci-dessous n'opèrent que sur des formats flottants. Historiquement, le 8087 était un coprocesseur séparé 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
<b>8087 Original x87</b>	-	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
<b>80287</b>	-	FSETPM
<b>80387</b>	-	FCOS, FLDENVD, FNSAVED, FNSTENV, FPREM1, FRSTORD, FSAVED, FSIN, FSINCOS, FSTENV, FUCOM, FUCOMP, FUCOMPP
<b>Pentium pro</b>	-	FCMOVB, FCMOVBE, FCMOVE, FCMOVNB, FCMOVNBE, FCMOVNE, FCMOVNU, FCMOVU, FCOMI, FCOMIP, FUCOMI, FUCOMIP, FXRSTOR, FXSAVE
<b>Pentium 4</b>	<b>SSE3</b>	FISTTP

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

- **MMX** : MultiMedia eXtensions
- **SSE** : Streaming SIMD Extensions
- **AVX** : Advanced Vector Extensions
- **AES** : Advanced Encryption Standard

CPU Architecture	Nom extension	Instructions	
Pentium MMX	MMX	EMMS, MOVD, MOVQ, PACKSSDW, PACKSSWB, PACKUSWB, PADDB, PADDD, PADDSB, PADDSD, 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, MOVHLP, MOVHPS, MOVLHP, 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, PMAWS, PMAWB, PMINSW, PMINUB, PMOVMASKB, 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, CVTTPS2SI, 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, PSHUFW, PSHUFLW, PSHUFD, PSLLDQ, PSRLDQ, PUNPCKHQDQ, PUNPCKLQDQ
	SSE3	Float Inst.	ADDSDPD, ADDSDPBS, HADDPD, HADDPBS, HSUBPD, HSUBPBS, MOVDDUP, MOVSHDUP, MOVSLDUP

Les instructions et opérandes usuellement manipulées par grand nombre de CPU sur le marché sont dites scalaires. Nous parlerons de **processeur scalaire** (PIC18 de Microchip, 8051 de Intel, AVR de Atmel...). Par exemple sur 8086 de Intel , scalaire + scalaire = scalaire :

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

A titre indicatif, les instructions MMX, SSE, AVX, AES ... sont dites 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
```

Cette instruction vectorielle peut notamment être très intéressante pour des applications de traitement numérique du signal : **dpps signifie dot product packet single**, soit produit scalaire sur un paquet de données au format flottant en simple précision. 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 DPPS <i>xmm1</i> , <i>xmm2/m128</i> , <i>imm8</i>	RMI	V/V	SSE4_1	Selectively multiply packed SP floating-point values from <i>xmm1</i> with packed SP floating-point values from <i>xmm2</i> , add and selectively store the packed SP floating-point values or zero values to <i>xmm1</i> .

<http://www.intel.com>

## Etudions 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]  $\leftarrow$  DEST[31:0] \* SRC[31:0];  
ELSE Temp1[31:0]  $\leftarrow$  +0.0; FI;

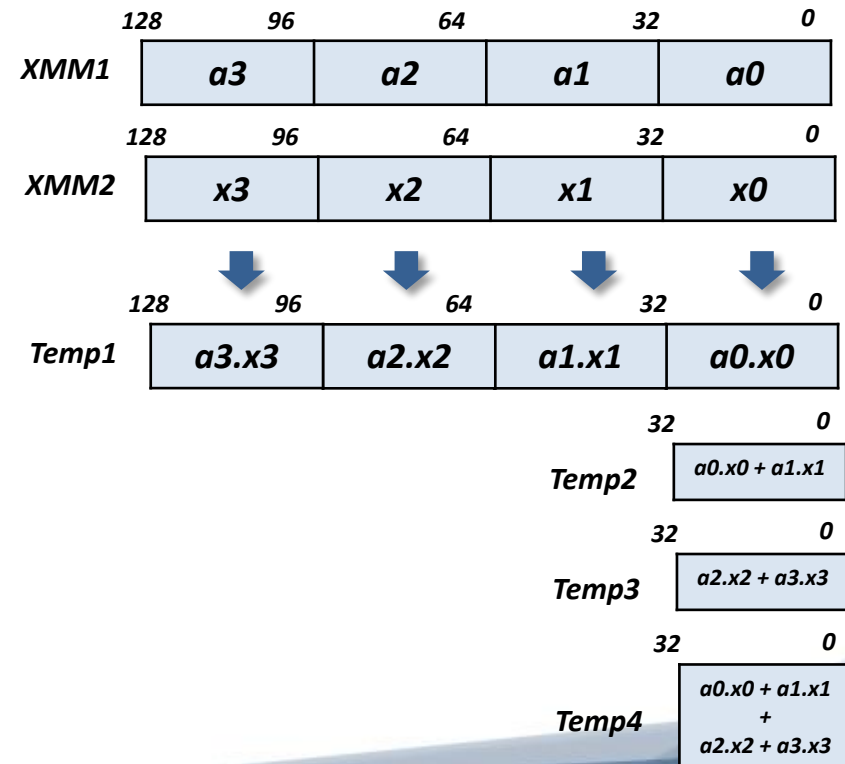
IF (imm8[5] = 1)  
THEN Temp1[63:32]  $\leftarrow$  DEST[63:32] \* SRC[63:32];  
ELSE Temp1[63:32]  $\leftarrow$  +0.0; FI;

IF (imm8[6] = 1)  
THEN Temp1[95:64]  $\leftarrow$  DEST[95:64] \* SRC[95:64];  
ELSE Temp1[95:64]  $\leftarrow$  +0.0; FI;

IF (imm8[7] = 1)  
THEN Temp1[127:96]  $\leftarrow$  DEST[127:96] \* SRC[127:96];  
ELSE Temp1[127:96]  $\leftarrow$  +0.0; FI;

Temp2[31:0]  $\leftarrow$  Temp1[31:0] + Temp1[63:32];  
Temp3[31:0]  $\leftarrow$  Temp1[95:64] + Temp1[127:96];  
Temp4[31:0]  $\leftarrow$  Temp2[31:0] + Temp3[31:0];

XMMi (i = 0 à 15 with Intel 64)  
128bits General Purpose Registers  
for **SIMD Execution Units**

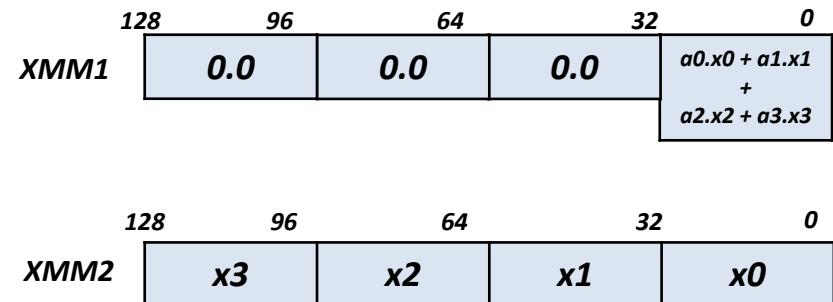
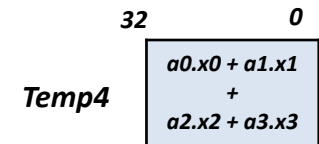




## Etudions un exemple d'exécution de l'instruction dpps :

**dpps 0xF1, %xmm2,%xmm1**

*XMMi (i = 0 à 15 with Intel 64)  
128bits General Purpose Registers  
for **SIMD Execution Units***



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)**

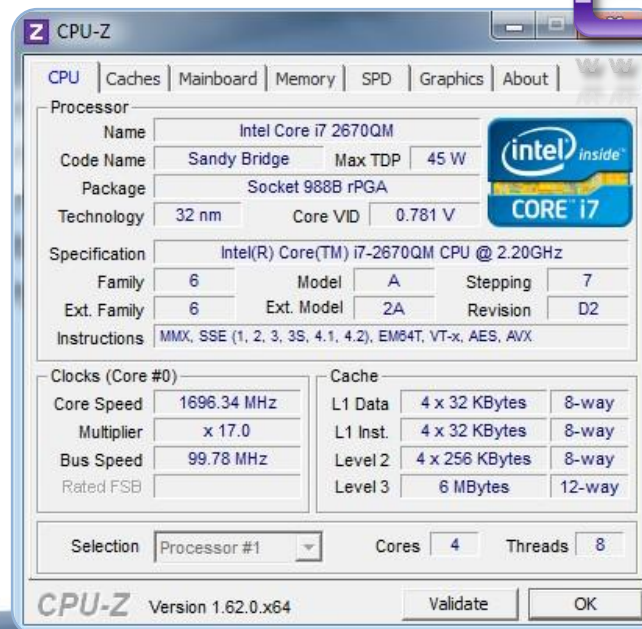
DEST[127:0] ← DP\_Primitive(SRC1[127:0], SRC2[127:0]);  
DEST[VLMAX-1:128] (Unmodified)

<http://www.intel.com>

*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..).*

<b>CPU Architecture</b>	<b>Nom extension</b>	<b>Instructions</b>
<b>Core2</b>	<b>SSSE3</b>	<i>PSIGNW, PSIGND, PSIGNB, PSHUFB, PMULHRSW, PMADDUBSW, PHSUBW, PHSUBSW, PHSUBD, PHADDW, PHADDSW, PHADDD, PALIGNR, PABSW, PABSD, PABS</i>
<b>Core2 (45nm)</b>	<b>SSE4.1</b>	<i>MPSADBW, PHMINPOSUW, PMULLD, PMULDQ, <b>DPPS</b>, DPPD, BLENDPS, BLENDPD, BLENDVPS, BLENDVPD, PBLENDVB, PBLENDW, PMINSB, PMAXS, PMINUW, PMAXUW, PMINUD, PMAXUD, PMINS, PMAXSD, ROUNDPS, ROUNDSS, ROUNDPD, ROUNDD, INSERTPS, PINSRB, PINSRD/PINSRQ, EXTRACTPS, PEXTRB, PEXTRW, PEXTRD/PEXTRQ, PMOVXSBW, PMOVZXBW, PMOVXBD, PMOVZBXD, PMOVXSBQ, PMOVZXBQ, PMOVXSWD, PMOVZXWD, PMOVXWQ, PMOVZXWQ, PMOVXDQ, PMOVZXDQ, PTEST, PCMPSEQ, PACKUSDW, MOVNTDQ</i>
<b>Nehalem</b>	<b>SSE4.2</b>	<i>CRC32, PCMPSTRI, PCMPSTRM, PCMPISTRI, PCMPISTRM, PCMPGTQ</i>
<b>Sandy Bridge</b>	<b>AVX</b>	<i>VFMADDPD, VFMADDPS, VFMADDSD, VFMADDSS, VFMADDSPD, VFMADDSPS, VFMSUBADDPD, VFMSUBADDPS, VFMSUBPD, VFMSUBPS, VFMSUBSD, VFMSUBSS, VFNMADDPD, VFNMADDPS, VFNMADDSD, VFNMADDSS, VFNMSUBPD, VFNMSUBPS, VFNMSUBSD, VFNMSUBSS</i>
<b>Nehalem</b>	<b>AES</b>	<i>AESEN, AESENCLAST, AESDEC, AESDECLAST, AESKEYGENASSIST, AESIMC</i>

*L'instruction CUID arrivée avec l'architecture Pentium permet de récupérer très facilement toutes les informations relatives à l'architecture matérielle du GPP (CPU's, Caches, adressage virtuel..). L'utilitaire libre CPU-Z utilise notamment ce registre pour retourner des informations sur l'architecture :*



CPUID

*Sous Linux, vous pouvez également consulter le fichier **/proc/cpuinfo** listant les informations retournées par l'instruction CPUID :*

```
vmlinux@vmlinux: ~  
vmlinux@vmlinux:~$ cat /proc/cpuinfo | more  
processor           : 0  
vendor_id           : GenuineIntel  
cpu family          : 6  
model               : 42  
model name          : Intel(R) Core(TM) i7-2670QM CPU @ 2.20GHz  
stepping            : 7  
cpu MHz              : 2107.531  
cache size          : 6144 KB  
physical id         : 0  
siblings             : 4  
core id             : 0  
cpu cores           : 4  
apicid              : 0  
initial apicid      : 0  
fdiv_bug            : no  
hlt_bug             : no  
f00f_bug            : no  
coma_bug            : no  
fpu                 : yes  
fpu_exception       : yes  
cpuid level         : 5  
wp                 : yes  
flags               : fpu vme de pse tsc msr pae mce cx8 apic se  
pat pse36 clflush  mmx fxsr sse sse2 ht syscall nx rdtscp lm  
e3 lahf_lm  
bogomips            : 4215.06  
clflush size        : 64  
cache_alignment     : 64
```





*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>



intel Menu Find Content Communities What can we help you find today? Sign In

<More on Intel.com Tagged As Architecture & Silicon, Software Developers

Recommend 231 +1 92 More

## Intel® 64 and IA-32 Architectures Software Developer Manuals

Combined Volume Set of Intel® 64 and IA-32 Architectures Software Developer's Manuals

Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 1, 2A, 2B, 2C, 3A, 3B, and 3C	This document contains the following: Volume 1: Describes the architecture and programming environment of processors supporting IA-32 and Intel 64 Architectures. Volume 2: Includes the full Instruction Set Reference, A-Z, in one volume. Describes the format of the instruction and provides reference pages for instructions. Volume 3: Includes the full System Programming Guide, Parts 1, 2, and 3, in one volume. Describes the operating-system support environment of Intel 64 and IA-32 Architectures, including: memory management, protection, task management, interrupt and exception handling, multi-processor support, thermal and power management features, debugging, performance monitoring, system management mode, VMX instructions, and Intel® Virtualization Technology (Intel® VT).
Intel® 64 and IA-32 Architectures Software Developer's Manual Documentation Changes	Describes bug fixes made to the Intel 64 and IA-32 Architectures Software Developer's Manual between versions. NOTE: This Change Document applies to all Intel 64 and IA-32 Architectures Software Developer's Manual sets (combined volume set, 3 volume set and 7 volume set).





***Merci de votre attention !***