# Programmation

* Solution programmée
  + Programme :
    - opérations élémentaires
    - données
  + Instruction élémentaire :
    - une opération
    - des opérandes sources
    - un opérande destination
  + Opérandes :
    - dans la mémoire ou dans un registre
    - explicites (dans l’instruction) ou implicites (l’instruction agit toujours sur cette même opérande
  + Recherche d’instruction, décodage, accès mémoire, exécution, stockage si besoin.
* Architecture
  + Modèle HARVARD
    - Mémoires séparées
    - Microcontrôleurs
  + Modèle VON NEUMANN (PRINCETON)
    - Mémoire unique
    - Microprocesseurs
* Système complet
  + Structure logique :
    - Processeur relié aux bus de communication
    - RAM, ROM
    - Contrôleur de disque lié aux périphériques de stockage
    - Contrôleurs réseau, USB, claviers/souris liés aux périphériques externes et périphériques d’interaction
  + 3 composants essentiels pour mémoriser, exécuter, communiquer.
  + Structure physique :
    - Processeur lié à la mémoire cache (RAM) (bus interne)
    - Ram liée au Northbridge (bus processeur)
    - Northbridge lié au Southbridge
    - Souhbridge lié aux contrôleurs (bus PCI)
    - Northbridge lié au bus mémoire (=> Mémoire centrale (RAM) et BIOS (ROM)) et à la vidéo (bus AGP)

# Mémoire

* Hiérarchie
  + Programmes et données stockés.
  + RAM, ROM, cache.
  + Autres mémoires (disques durs, flash …) :
    - Grandes quantités de données
    - Pas d’accès direct pour le processeur
  + Plus la vitesse d’accès est grande, plus la capacité est faible et le coût élevé.
* Organisation
  + Séquences d’octets qui ont chacun une adresse unique codée sur n bits. L’adresse d’un octet est donc entre 0 et 2n-1.
  + Groupe d’octets : adresse du groupe = adresse du premier octet.
  + Ordres « Little Endian » ou « Big Endian ». En premier : octet de poids faible ou de poids fort.

# Processeur

* Machine à accumulateur
  + Registre interne qui sert d’opérande source et destination
* Machine à registres :
  + Plus d’accumulateurs !
* Machine à pile :
  + Toutes les opérations sont effectuées sur la pile
* Classes de machines
  + CISC :
    - Plus d’une opération élémentaire par instruction => code plus condensé, moins de mémoire pour stocker les programmes
    - Peu de registres
  + RISC :
    - Instructions simples => optimisation du matériel plus aisée
    - Plus de resgitres

# Famille x86 d’Intel

* Présentation
  + Little Endian
* Modes de fonctionnement
  + Réel ou protégé :
    - Réel à la mise sous tension, émulation d’un 8086, accès à 1Mo de mémoire seulement
    - Protégé, permet l’accès à tout l’espace d’adressage
  + Opérande 16 bits ou 32 bits (pour les plus récents).
* Déroulement d’une instruction
  + Lecture de l’instruction et incrémentation de EIP.
  + Décodage.
  + Calcul de l’adresse des opérandes.
  + Lecture des opérandes.
  + Opération (UAL).
  + Positionnement des indicateurs (flags).
  + Ecriture du résultat.

# Outils

Assembleur …

Editeur de liens

* + L’assemblage (ou la compilation) d’un fichier source produit un fichier objet.
  + L’éditeur de liens prend en entrée les fichiers objets.
  + Il les rassemble et décide des adresses mémoires définitives des données et instructions

# Registres

* Usage général
  + EAX, EBX, ECX, EDX : 32 bits
  + AX, BX, CX, DX : 16 bits
  + AH, AL, BH, BL, CH, CL : 8 bits
* Index
  + ESI, EDI : 32 bits
  + SI, DI : 16 bits
* Pointeurs
  + ESP : pointeur de pile
  + EBP : pour manipuler des données sur la pile
  + EIP : pointeur d’instructions
* Drapeaux
  + CF : retenue
  + ZF : zéro (1 si nul)
  + SF : signe (1 si négatif)
  + OF : débordement
* Segments
  + Utilisés pour la gestion mémoire
  + Configurer l’espace d’adressage dédié aux éléments du programmes :
    - CS pour instructions
    - DS pour les données
    - SS pour la pile
    - ES, FS, GS pour définir des espace supplémentaires
  + Cf. Gestion de la mémoire

# Encodage des instructions : Cf. Encodage des instructions.pdf

# Sous-programmes

* Propriétés
  + Réentrant : peut-être interrompu et appelé par le programme qui l’a interrompu.
  + Récursif : il s’appelle lui-même. Doit également être réentrant.
  + Arguments d’entrée et de sortie. Informations variables en fonction du contexte.
* Appels de sous-programmes
  + CALL : EIP ← EIP+1, et on PUSH cette nouvelle valeur de EIP. Ensuite, EIP prend la valeur de la première instruction du sous-programme.
  + RET : opération inverse de CALL. Dépile l’adresse de retour et l’affecte à EIP. Il faut donc s’assurer que ESP pointe bien sur la valeur de l’adresse retour !
  + Remarque : la pile n’a pas une taille infinie. Attention aux sous programmes imbriqués, notamment dans le cas de la récursivité.
  + Attention aux registres ! Sauvegarde au début du sous-programme.
* Passage d’arguments
  + Par la mémoire :
    - Les arguments sont stockés dans la mémoire à une adresse connue. Le programme va chercher les valeurs qui y sont pendant l’exécution du sous programme. Le programme principal doit donc remplir les cases mémoires à ces adresses connues.
    - Inconvénients :
      * Sous-programme non réentrant
      * Caractère fixe => manque de souplesse
      * Finalement c’est comme utiliser des variables globales au sein d’une fonction !!!
  + Par registres :
    - Avantages : appels plus rapides, moins d’instructions et d’appels mémoires, sous programme plus rapide
    - Inconvénient : nombre d’arguments limités aux nombre de registres
  + Par la pile :
    - Arguments positionnés sur la pile dans un ordre déterminé
    - Lecture directe dans la pile en utilisant un déplacement par rapport à ESP
    - ESP peut varier !!!
      * Difficile d’en prévoir l’évolution.
      * Solution : cadre de pile :
        + Stocker dans un registre la valeur du pointeur après l’appel au sous-programme. Généralement, EBP.
        + PUSH EBP // MOV EBP,ESP
    - Suppression des arguments :
      * Il faut supprimer les arguments inutiles après l’exécution. Qui le fait ?
        + Le programme …
        + … ou le sous-programme
      * RET n, ou n est le nombre d’octets à supprimer de la pile, permet au sous-programme de le faire
      * Incrémenter ESP permet au programme de le faire : ADD ESP, n

# Mémoire physique

* Processeur connecté grâce aux bus
  + D’adresses : taille impose celle de la plus grande adresse
  + De données : taille impose la quantité de données échangeables en une fois
  + Signaux de contrôle et de synchronisation
* Transferts : sélection de bus
  + Un signal par octet du bus de données
    - BLE actif pour valider D7…0
    - BHE actif pour valider D15…8
  + Bit d’adresse A0 représenté par le couple BLE/BHE
* Donnée alignée : son adresse est multiple de sa taille
* Espace d’adresage physique : pas toujours entièrement occupé

# Gestion de la mémoire

* Programme : image exécutable stockée en mémoire secondaire
* Processus : matérialisation d’un programme et de ses données quand ils sont chargés en mémoire principale
* Espace d’adressage logique associé à l’ensemble des adresses physiques correspondant au bloc de mémoire principale alloué au processus
* Adressage réel
  + Les instructions référencent les données au moyen de leur adresse
  + Il faut que les adresses logiques des différents processus soient différentes
  + Chaque programme doit connaître à l’avance sa place définitive en mémoire
  + Allocation statique
* Adressage moderne
  + Chaque processus est créé avec un espace d’adressage vide très étendu
  + Adresses logiques identiques dans plusieurs processus doivent être associées à des adresses physiques différentes
  + Système d’exploitation : gestionnaire de mémoire
    - Abstraction de la mémoire principale : chaque processus a l’illusion de disposer d’un grand espace d’adressage contigu
    - Allocation de blocs de mémoire principale à chaque processus
    - Isolation des processus entre eux
  + L’OS est assisté par la gestion de mémoire du processeur
  + Relogement dynamique
    - Gestionnaire de mémoire peut charger n’importe où un processus sans modifier le programme
    - Adresses absolues des programmes sont les mêmes
    - Adresses des instructions, données, pile, sont relatives au début d’adressage du processus ; transformées en adresses physiques par le processeur au cours de l’exécution ⇒ relogement dynamique
  + Registres de relogement multiples
    - Une adresse de base ≠ pour
      * Code
      * Données
      * Pile
  + Vérification des accès (protection) (contrôle bornes de l’adresse relative par rapport à une limite maximale)

# Mémoire virtuelle

* Concept abstrait : abstraction de la mémoire formée des espaces d’adresses logiques de tous les processus
* Espace d’adressage virtuel peut être identique pour tous les processus
* Gérée par le gestionnaire de mémoire
* Adresse virtuelle : point du vue du processus
* Translation d’adresse : conversion en adresse physique par l’unité de gestion mémoire du processeur (MMU)
* Deux mécanismes d’implémentation de la mémoire virtuelle :
  + Segmentation
    - Chaque processus est segmenté :
      * Segment de code
      * Segment de données
      * Segment de pile
    - Adresse linéaire = adresse virtuelle + adresse linéaire du début du segment
    - Existence d’une table de description (accessible par l’OS) où sont stockées
      * Adresse physique du début de chaque segment
      * Longueur de segment
      * D’autres informations
    - Cette table est stockée en mémoire physique
    - Chaque entrée dans la table représente un segment différent : descripteur de segment
    - Son index dans la table permet de retrouver un descripteur : sélecteur de segment
    - Avantages
      * Chaque segment occupe exactement la quantité de mémoire physique nécessaire
      * Tables de taille réduite (une entrée par segment)
    - Inconvénients
      * Fragmentation externe
      * Mécanisme d’allocation mémoire inefficace (recherche linéaire, compactage)
    - On peut spécifier explicitement le segment
      * MOV CS: [6000h], AX
      * MOV SS: [EBX + ESI \* 2 + 4], EAX
      * Substitution de segment
  + Pagination
    - Construire une table de correspondance adresse virtuelle ⇔ adresse physique
    - Avantage : chaque adresse virtuelle peut être à un emplacement différent en mémoire centrale. Une adresse virtuelle non utilisée peut être simplement ignorée (pas de correspondance)
    - Inconvénient : taille de la table de correspondance
    - Espace d’adressage virtuel divisé en zones contiguës de taille fixe (4ko en IA32) : pages
    - Idem avec la mémoire physique (4096 octets en IA32) : cadres de pages
    - Table des pages :
      * Une table par processus
      * Gérée par l’OS
    - Problème : limiter la taille des tables
    - Solution : diviser l’espace d’adressage virtuel en « groupes de pages », les groupes en groupes, … : plusieurs niveaux de table
    - Ex IA32 :
      * 1024 tables
      * Seules les tables utilisées effectivement sont allouées en mémoire principale
      * Une table principale : répertoire des pages
    - Translation : opération coûteuse
    - Principe de localité : il y a de fortes chances que l’accès mémoire suivant ait lieu à une adresse linéaire proche
    - TLB : mémoire associative interne pour enregistrer les dernières conversions
    - Avantages :
      * Allocation aisée
      * Pas de fragmentation externe
    - Inconvénients
      * Fragmentation interne
      * Plusieurs accès pour un accès mémoire ! (n pour les tables, 1 pour la donnée)
      * Taille des tables
  + Segmentation et pagination peuvent être mêlées
  + Récapitulation sur les adresses :
    - Adresses virtuelles : manipulées par le processus et le programmeur
    - Adresses linéaires : point de vue de l’espace total d’adressage du processeur — adresses physiques si pagination inactive
    - Adresses réelles ou physiques : vues par le matériel

# Présentation

* Temps d’exécution d’un programme pour un processeur
  + Texe = N \* CPI \* 1\F
    - N : nombre d’instructions
    - CPI : nombre de cycles d’horloge par instruction
    - F : fréquence d’horloge
  + Objectif : minimiser Texe
    - Minimiser N : optimiser les instructions
    - Minimiser CPI : utiliser le cache
    - Minimiser 1/F : augmenter la fréquence d’horloge

# Cache Memory

* Direct Mapped Cache
  + Advantages :
    - Simple
    - Fast
  + Disadvantage :
    - Mapping is fixed !!!
* Associative Caches