Architecture et Système

Stefan Schwoon

Cours L3, 2014/15, ENS Cachan

Rappels

Quelques éléments qu'on a pu construire à partir des transistors (et une horloge):

- fonctions arithmétiques et logiques
- multiplexeur, décodeur
- mémoire

On va discuter comment réaliser un ordinateur simple avec ces éléments.

Mémoire vive et morte

En parlant de *mémoire* (sans stockage en masse comme les disques durs), on distingue notamment deux types :

Mémoire vive

Mémoire en mode lecture ou écriture (p.ex. des bascules)

volatile (contenu perdu sans alimentation électrique)

utilisé pour faire des calculs, stocker des données

en anglais: RAM (random-access memory)

Mémoire morte

Mémoire en mode lecture seulement (pas de moyen pour l'ordinateur ou le programmeur d'en modifier)

durable (même sans alimentation électrique)

utilisé pour assurer le fonctionnement de l'ordinateur (microprogrammation, tables mathématiques pré-calculées)

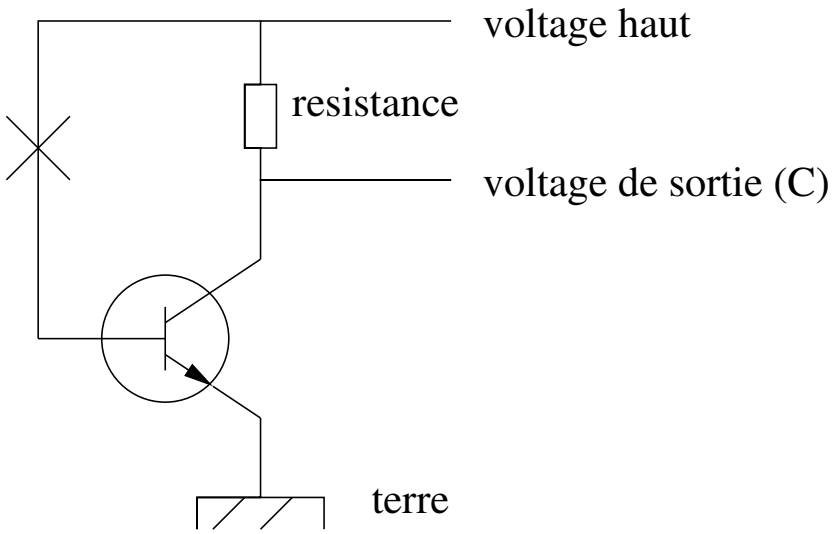
en anglais: ROM (read-only memory)

Types de mémoire morte

Fa**c**con la plus simple de produire un ROM :

prendre une série de transistors, chacun représentera un bit

physiquement interrompre le fil à la position X pour donner 1, sinon 0.



→ “programmable” une seul fois

→ appelé PROM quand la programmation se fait après la production initiale

EPROM

EPROM = Erasable programmable ROM
et EEPROM = electrically erasable PROM

Consiste des transistors avec une porte supplémentaire (“floating gate”) qui accède à une couche entre la base et les autres élements du transistor.

Le “floating gate” est programmable avec des charges électriques supérieur à ceux utilisé pendant l’opération normale de l’ordinateur.

$\rightarrow$ reprogrammable par l’extérieur, mode lecteur seulement pour l’ordinateur

Programmable logic devices (PLA)

n bits d'entrée, m bits de sortie

tout les bits d'entrée disponible aussi en négation

Partie ET: $k \leq 2^n$ lignes, chacune choisit une conjonction d'entrées

Partie OU: m colonnes, tout bit de sortie est formé par la disjonction d'un sous-ensemble des k lignes

programmation consiste en choisissant les conjonctions et disjonctions

Field-programmable gate arrays (FPGA)

plusieurs lignes de *cellules logiques*

chaque cellule peut réaliser des fonctions diverses arithmétiques et logiques, sélectable par multiplexeur

programmation consiste en choisissant la fonction de chaque cellule et les connections entre celles-ci

Architecture d'un ordinateur simple

Les élements:

processeur (CPU), consiste d'une unité de contrôle (CU), unité arithmétique-logique (ALU), unité de microprogrammation et horloge

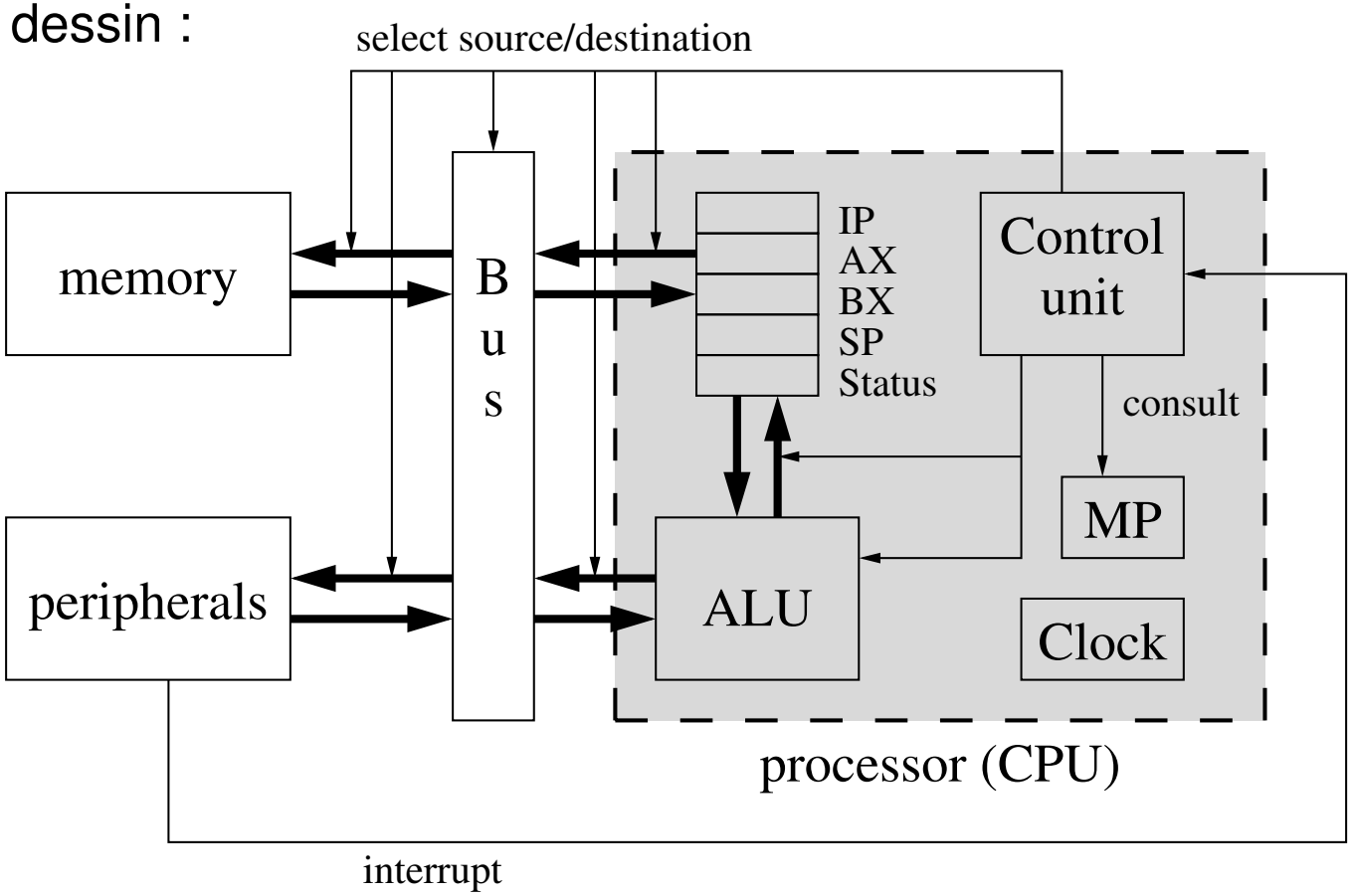
mémoire principale

les p'eriphériques (fournissent des entrées/sorties pour les données)

le bus qui transfert les données entre les composants

Architecture

Un premier dessin :



Mémoire principal

Pour stocker données et instructions (architecture dite “von Neumann”)

Organisée en 2^n mots de m bits, les mots stockés à des *adresses* $0..2^n - 1$

Valeur typique pour m: 8; pour n: 30..34 (= 1..16 GB)

Réalisable, par exemple, avec des bascules D (details à discuter plus tard)

Échanges avec le processeur :

lecture : multiplexeur pour choisir un mot à une adresse précise, transfert au bus

obtenir un mot et une adresse du bus, utiliseur décodeur pour l’écrire à la bonne destination

Le bus

Pour connecter CPU, mémoire, périphériques: tous composants peuvent lire et écrire

Utiliseur des multiplexeurs et décodeurs pour sélectionner source et destination du transfert

Un transfert à la fois (on peut pas obtenir deux mots de la mémoire au même temps, par exemple).

Le processeur central (CPU)

Contrôle toutes les opérations, avec les composants suivants:

registres: mèmoire de court terme pour les opérations actuelles

ALU: pour manipuler des données

control unit: pour coordonner les transferts, avec l'aide de l'horloge et l'unité de microprogrammation

Registres

Un registre stocke un mot (le mot peut être de taille différent des mots en mémoire, typiquement 32 ou aujourd'hui).

Il existe des registres pour des objectifs différents.

registres géneraux (accumulateur, AX, BX, . . . (eax, ebx, . . .): utilisé pour les calculs

compteur d'instructions: va pointer vers la prochaine instruction à exécuter dans la mémoire

registre de status: stock quelques bits pour indiquer des informations sur le succés ou non d'une opération (comparaison, division par zéro, . . .)

pointeur de pile: indique le sommet d'une pile qui sauvegarde des valeurs de certains registres, adresse de retour pour les sous-routines, etc

Unité de contrôle (CU)

Travaille dans des phases indiqués par l'horloge :

Chaque phase réalise une opération nécessaire pour exécuter une instruction:

Dans chaque phase, le CU doit organiser les transferts de données:

- donner les bons bits de sélection au bus
- transferer les donées entre ALU et registres
- réagir aux périquériques

Quelques phases typiques: obtenir une instruction, décoder une opération, exécuter une opération (en plusieurs phases)

Un ROM (unité de microprogrammation) peut servir pour fournir des signaux à donner aux autres élements

Phases

Phase 1 (Instruction fetch):

Laisser le bus utiliseur la valeur d'IP comme sélecteur d'adresse, transferer au CPU

Phase 2 (Decode):

Consulter unité de microprogrammation pour organiser les transferts néseccaires pour l'opération

Phases 3, 4, . . . (Execute):

En utilisant le MU, organiser les bons transferts et manipulations dans chaque phase.

Des instructions différents peuvent nécessiter des differents temps.

Interrupts

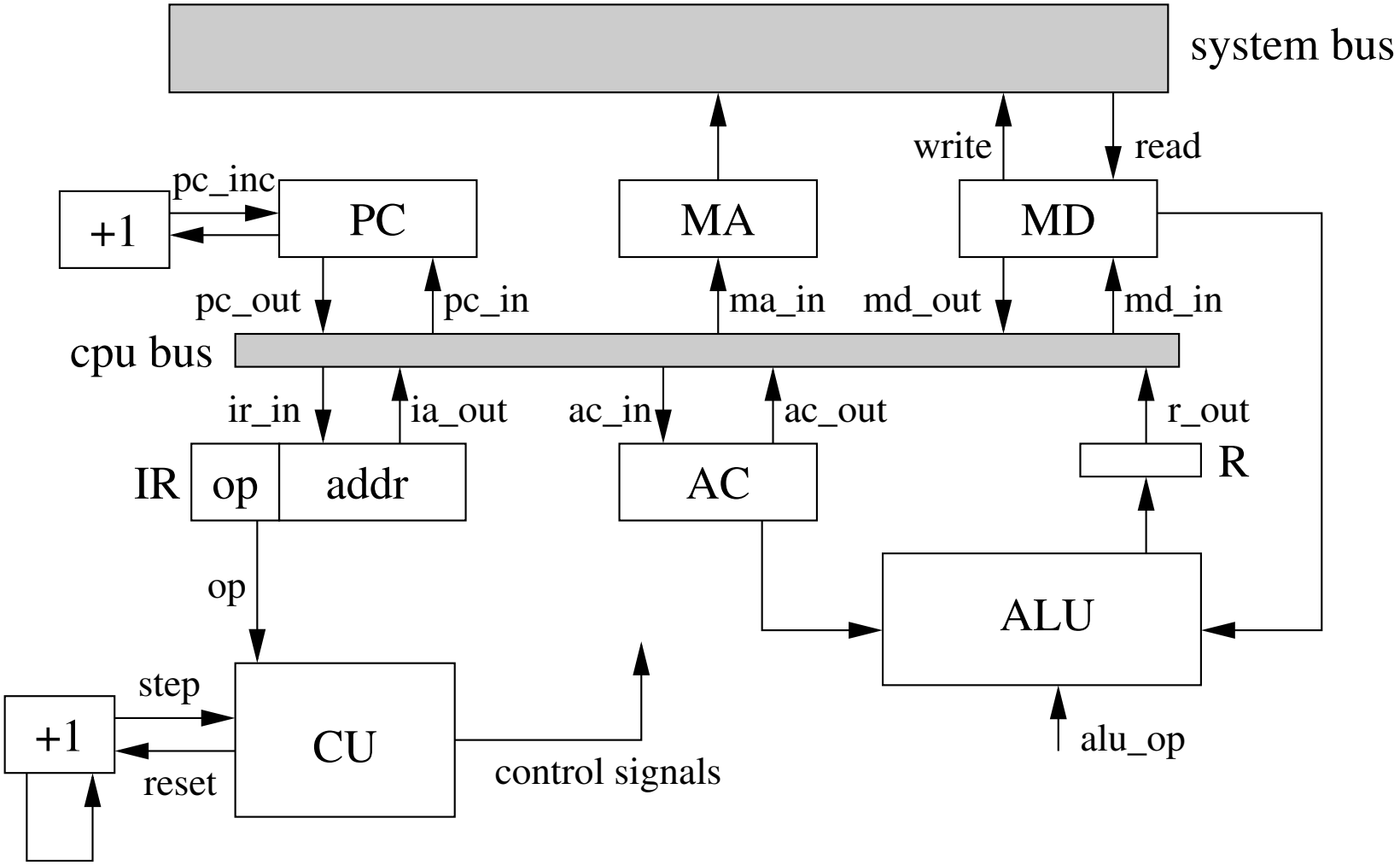
When a peripheral needs to exchange data with the CPU, it raises an interrupt signal at the control unit.

When the CU is ready to execute the next instruction, it first checks whether at least one interrupt signal is present.

If so, it chooses the one with the highest priority.

The CPU then proceeds with an instruction that handles communication with the peripheral.

Sample CPU design



This design has a CPU-internal bus and a system bus.

MA/MD communicate with the memory via system bus.

IR: instruction register / MA: memory address / MD: memory data

Input/output to buses controlled by the named signals written next to them; these are provided by the CU.

Executing an operation

Let us regard two operations:

LOAD *addr*: load content of addr into accumulator

- phase 3: load *addr* into MA: (`ia_out`, `ma_in`)
- phase 4: fetch data from memory (`read`)
- phase 5: transfer to ALU, next op (`md_out`, `alu_in`, `reset`)

ADD *addr*: add content of addr to accumulator

- phase 3: load *addr* into MA: (`ia_out`, `ma_in`)
- phase 4: fetch data from memory (`read`)
- phase 5: perform addition (`alu_op=add`)
- phase 6: transfer to ALU, next op (`r_out`, `alu_in`, `reset`)