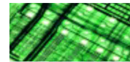
SETRE: Systèmes Embarqués Temps Réel

Tanguy Risset, Antoine Fraboulet
tanguy.risset@insa-lyon.fr
Lab CITI, INSA de Lyon

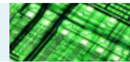
Plan du cours 1 (3H)

- architecture du MSP430
- Rappel d'architecture (processeurs embarqués)
- Outils de développement: Compilation et assembleur

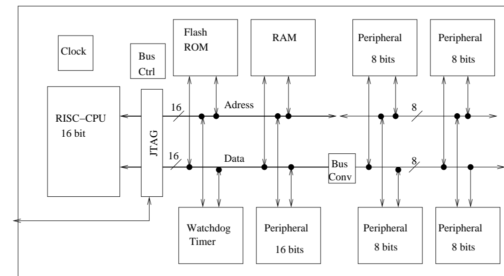


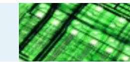
Références

- Documents du cours `http://www.setre.info/`
- The `mspgcc` toolchain: `http://mspgcc.sourceforge.net/`
- Notamment la doc `mspgcc` (*download* puis *documentation)*
- Doc du MSP430 (MSP430x14x family)
- Compilation:
- David Patterson , John Hennessy *Computer Organization and Design: the hardware software interface* Morgan-Kaufmann, 1998.
- Wayne Wolf Computers as Components: Principles of Embedded Computing System Design. Un survey assez complet de toutes les technologies utilisées
- Liens sur le site du cours système embarqués (Master Recherche): `http://citi.insa-lyon.fr/~trisset/cours/MasterWeb/`



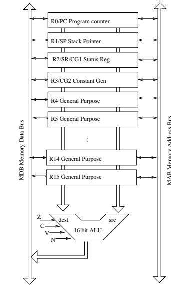
Architecture du MSP





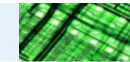
CPU RISC 16 bits

- 28 Instructions sur 16 bits
- 64 Ko de mémoire adressable
- Périphériques mappés en mémoire
- 16 registres 16 bits (r0-r16)
  - r0: PC (Program counter)
  - r1: SP (Stack pointeur)
  - r2: SR (status register)
  - r3: constante 0



R3: Status Register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							V	SCG1	SCG1	OS–COFF	CPU OFF	GIE	N	Z	GC



Periphérique intégrés

- timers
- contrôleur LCD
- mutliplieur cablé
- contrôleur de bus
- convertisseur analogique numérique, comparateur
- ports séries

Périphérique mappé en mémoire

- l'écriture à une certaine adresse est interprétée comme une communication avec le périphérique.
- Exemple : le multiplieur matériel
  - Accessible par des registres mappés entre les adresses `0x0130` et `0x013F`
  - Écriture à l'adresse `0x130`, positionne le premier opérande (unsigned mult)
  - Écriture à l'adresse `0x138`, positionne le deuxième opérande et lance le calcul
  - Le résultat est à l'adresse `0x013A`, sur 32 bits
- Les autres périphériques sont aussi accessibles par des registres mappé en mémoire: les SFR (Special Function Registers), en C:
  - écriture vers le périphérique: `SFR = valeur`
  - lecture des registres du périphérique: `variable = SFR`

Mapping mémoire

introduction au MSP 430
- Architecture
- Périphériques
- Chaîne de développement

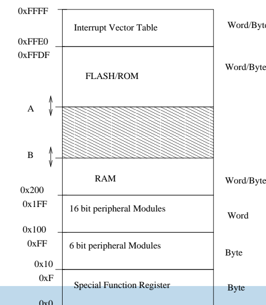
Processeurs embarqués

Outils GNU pour la compilation

Un compilateur pour l'embarqué: GCC

Pile d'exécution

- Interrupt Vector Table: indique les adresses des fonctions de gestion des interruption
- Lors du Boot, le MSP va lire l'adresse `0xFFFE`: handler du reset.
- Sur le MSP430F449:
  - `0x0` à `0x1FF`: périphériques
  - `0x200` à B=`0x9FF`: RAM (2Ko), Données et pile d'exécution
  - `0xC00` à `0xFFF`: Boot mem.
  - `0x1000` à `0x10FF`: byte info. mem.
  - A=`0x1100` à `0xFFFF`: ROM (60 Ko): code.



Chaîne de développement

introduction au MSP 430
- Architecture
- Périphériques
- Chaîne de développement

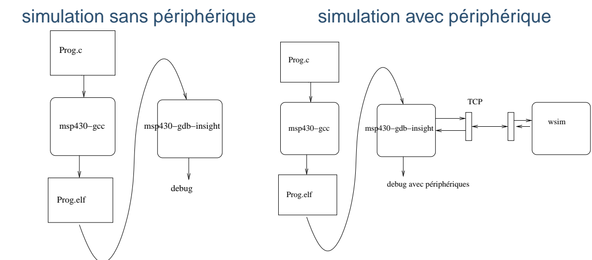
Processeurs embarqués

Outils GNU pour la compilation

Un compilateur pour l'embarqué: GCC

Pile d'exécution

simulation sans périphérique simulation avec périphérique



Exemple: multiplieur cablé

introduction au MSP 430
- Architecture
- Périphériques
- Chaîne de développement

Processeurs embarqués

Outils GNU pour la compilation

Un compilateur pour l'embarqué: GCC

Pile d'exécution


```
int main(void) {
    int i;
    int *p,*res;

    p=0x130;
    *p=2;
    p=0x138;
    *p=5;
    res=0x13A;
    i=*res;

    nop();
}
```

```
int main(void) {
    int i;
    int *p,*res;

    __asm__("mov #304, R4");
    __asm__("mov #2, @R4");
    // p=0x130;
    // *p=2;
    __asm__("mov #312, R4");
    __asm__("mov #5, @R4");
       //p=0x138;
       //*p=5;
    __asm__("mov #314, R4");
    __asm__("mov @R4, R5");
      //res=0x13A;
    i=*res;

    nop();
```


Chaîne de développement

introduction au MSP 430
- Architecture
- Périphériques
- Chaîne de développement

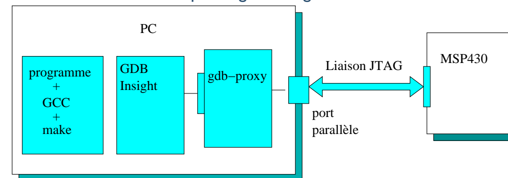
Processeurs embarqués

Outils GNU pour la compilation

Un compilateur pour l'embarqué: GCC

Pile d'exécution

- Plusieurs configurations
  - Programmation directe du micro-contrôleur
  - Simulation avec msp430-gdb-insight (sans les périphériques)
  - Simulation avec msp430-gdb-insight et wsim



Chaîne de développement

introduction au MSP 430
- Architecture
- Périphériques
- Chaîne de développement

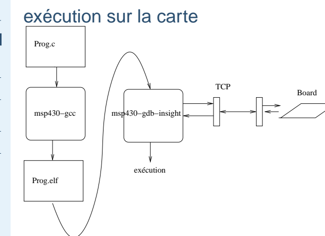
Processeurs embarqués

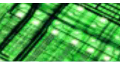
Outils GNU pour la compilation

Un compilateur pour l'embarqué: GCC

Pile d'exécution

exécution sur la carte





introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques

Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

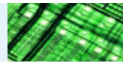
Processeurs embarqués

Part de marché

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques

Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Quel est le le microprocesseur le plus vendu ?
  - Réponse classique: "Le Pentium: 92% du marché"
- Faux!......
  - En fait les Pentium ne représentent que 2% des microprocesseurs vendus dans le monde.

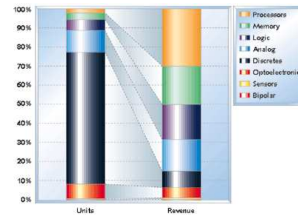


Contradiction ?

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques

Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Alors d'ou vient la position d'Intel (16% du marché des semi-conducteurs) ?
- processeurs: 2% du silicium, 30% des revenus

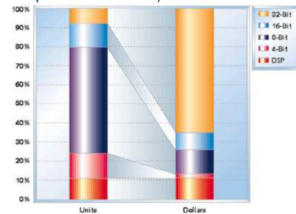


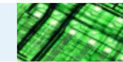
Et au sein des processeurs

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques

Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- 3 milliards de processeurs 8 bits vendus par an (8051, 6805 etc.)
- 32 bits (Pentium, Athlon, mais aussi PowerPC, 68000, MIPS, ARM etc.)
- La plupart (98%) sont embarqués (3 fois plus d'ARM vendus que de Pentium)



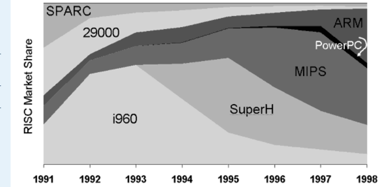


Variété des processeurs embarqués

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques

Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Embedded RISC Lead Swings Constantly



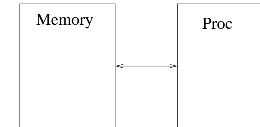
- Les applications sont plus variées que pour les ordinateurs
- Beaucoup de processeurs embarqués sont des processeurs de bureau qui n'ont pas percés (MIPS, 68K, SPARC, ARM, PowerPC)

Architecture "Von Neuman" ou "Princeton"

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques

Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

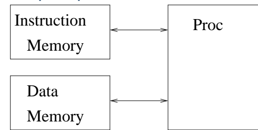
- La mémoire contient les données et les instructions
- L'unité centrale (CPU) charge les instructions depuis la mémoire.
- Un ensemble de registres aide le CPU:
  - Compteur d'instructions (Program counter: PC),
  - Registre d'instruction (Instruction register: IR)
  - Pointeur de pile (stack pointer: SP)
  - Registres à usage général (Accumulateur: A)



introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Architecture Harvard

- Données et instructions dans des mémoires séparées
- Autorise deux accès simultanés à la mémoire.
- Utilisé pour la plupart des DSP
  - meilleure bande passante
  - Performances plus prédictibles



introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Le jeu d'instruction

- Le *jeu d'instruction* (Instruction Set Architecture: ISA) a une importance capitale
  - Il détermine les instructions élémentaires exécutées par le CPU.
  - C'est un équilibre entre la complexité matérielle du CPU et la facilité d'exprimer les actions requises
  - On le représente de manière symbolique (ex: MSP, code sur 16 bits):

```
        mov r5,@r8 ; commentaire [R8]<-R5
lab:    ADD r4,r5  ; R5<-R5+R4
```

- Deux classes de jeux d'instructions:
  - CISC: Complex Instruction Set Computer
  - RISC: Reduce Instruction Set Computer

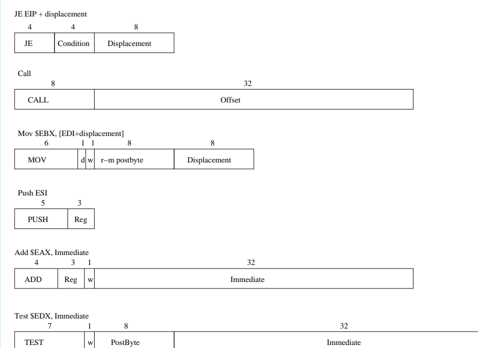
introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

CISC: Complex Instruction Set Computer

- Une instruction peut designer plusieurs opérations élémentaires.
  Ex: un load, une opération arithmétique et un store,
  Ex: calculer une interpolation linéaire de plusieurs valeurs en mémoire.
- Accélération par des mécanismes matériels complexes
- Grandes variations de taille et de temps d'exécution pour les instructions
- Résulte en un code compact mais complexe à générer.
- Vax, Motorola 68000, Intel x86/Pentium

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Exemple: instructions de l'ISA du Pentium



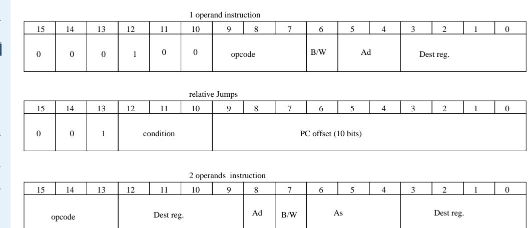
introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

RISC: Reduced Instruction Set Computer

- Petites instructions simples, toutes de même taille, ayant toutes (presque) le même temps d'exécution
- Pas d'instruction complexe
- Accélération en pipelinant l'exécution (entre 3 et 7 étages de pipeline pour une instruction) $\Rightarrow$ augmentation de la vitesse d'horloge
- Code plus simple à générer, mais moins compact
- Tous les microprocesseurs modernes utilisent ce paradigme: SPARC, MIPS, ARM, PowerPC, etc.

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Exemple: instructions de l'ISA du MSP

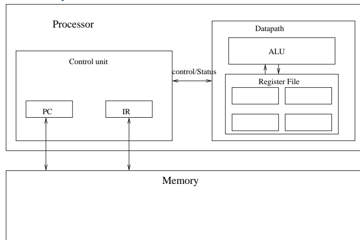


Exemples:

- `PUSB.B R4`
- `JNE -56`
- `ADD.W R4,R4`

Le CPU

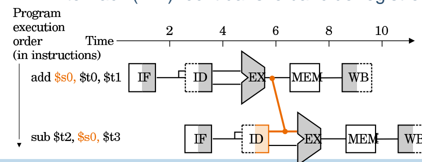
- L'unité de contrôle configure le chemin de donnée suivant l'instruction à exécuter.
- L'exécution d'une instruction est décomposée en plusieurs phases d'un cycle.



Le pipeline RISC: exemple du MIPS

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

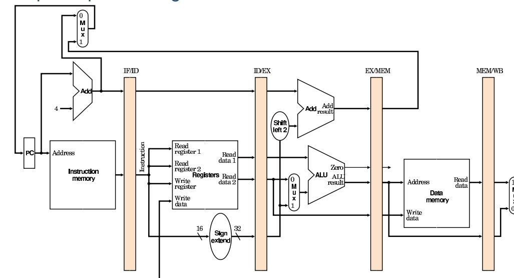
- Le pipeline dépend de l'architecture, pour le MIPS:
  - Instruction Fetch (IF, Fetch): charge l'instruction dans l'IR
  - Instruction Decode (ID, Decode): décode l'instruction et met en place le contrôle du chemin de donnée
  - Execute (Ex): exécute le calcul dans le chemin de donnée.
  - Memory access (Mem): accède la mémoire
  - Write Back (WB): écrit dans le banc de registre



Le pipeline RISC: exemple du MIPS

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

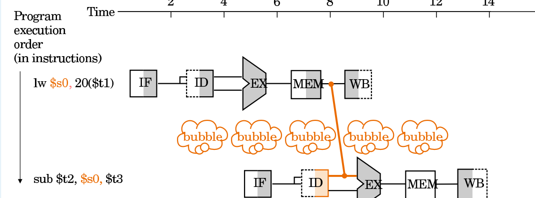
- Physiquement, l'architecture du processeur est organisée en calculs combinatoires pour chaque étape de pipeline, séparés par des registres.



Le pipeline RISC: exemple du MIPS

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

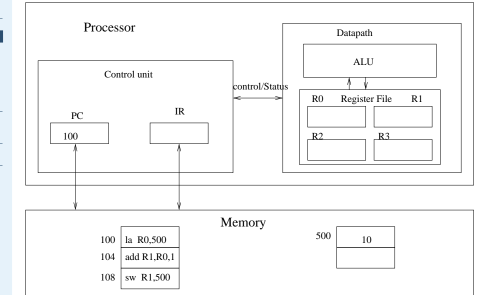
- Lorsque l'instruction suivante ne peut pas être exécutée tout de suite, cela crée une "bulle".
- Par exemple une addition utilisant un registre qui vient d'être chargé doit être retardé d'un cycle.



Exemple d'exécution sans pipeline

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

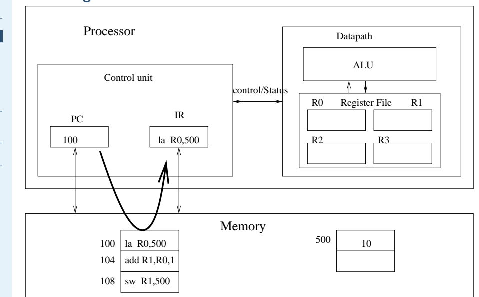
- Avant le début de l'exécution de l'instruction à l'adresse 100



cycle 1

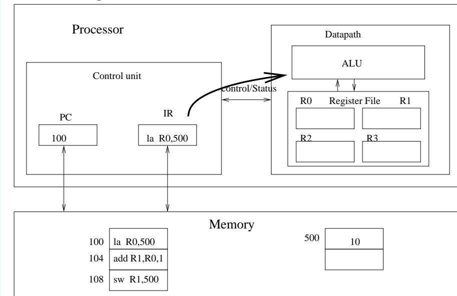
introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Chargement de l'instruction



cycle 2

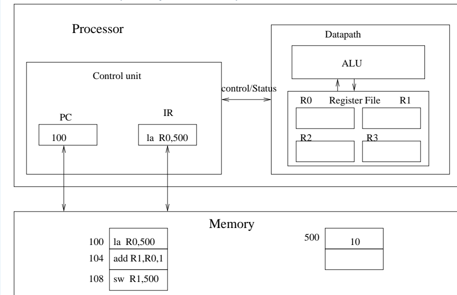
- Décodage de l'instruction



introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

cycle 3

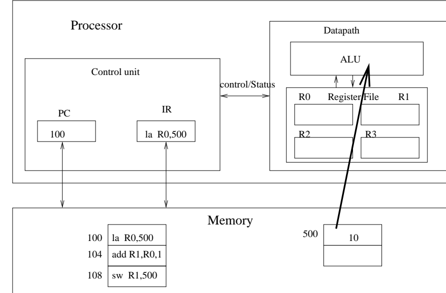
- Exécution (rien pour load)



introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

cycle 4

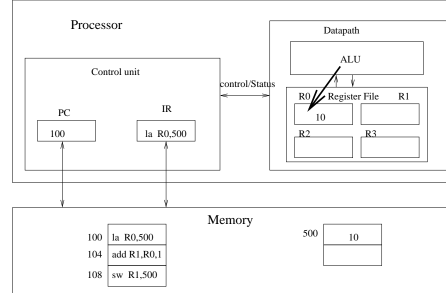
- Accès mémoire



introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

cycle 5

- Write Back



introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

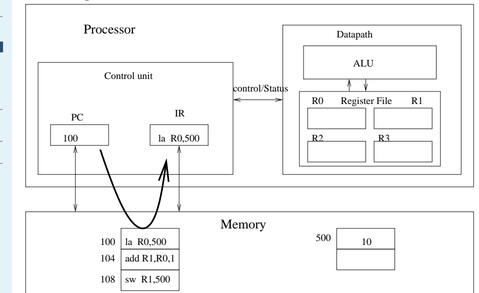
Bilan architecture pipelinée

- Exécution non pipelinée:
  - 5 cycles pour exécuter une instruction
  - $\Rightarrow$ 15 cycles pour 3 instructions.

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

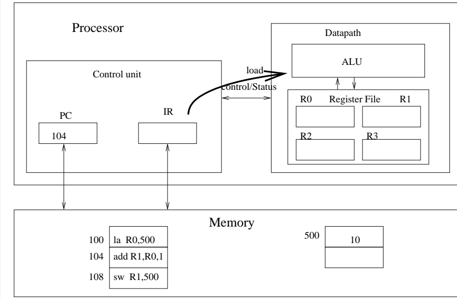
Exemple d'exécution avec pipeline

- Chargement de l'instruction load



cycle 2

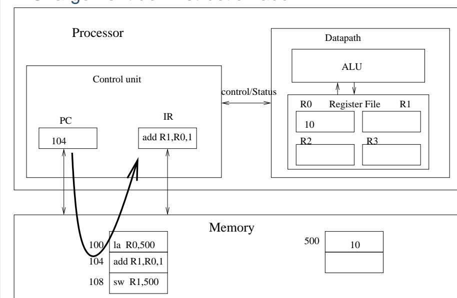
- Décodage de l'instruction load
- et Chargement de Rien (bulle car l'instruction suivante retardée)



cycle 3

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

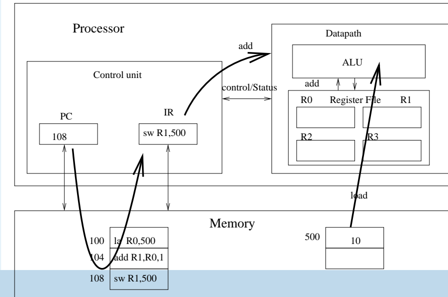
- Exécution de l'instruction load (rien)
- Décodage de rien
- Chargement de l'instruction add



cycle 4

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

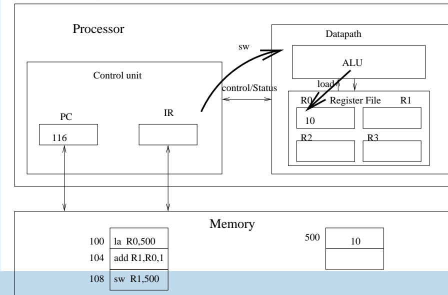
- Accès mémoire de l'instruction load
- Exécution de rien
- Décodage de l'instruction add
- Chargement de l'instruction store



cycle 5

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

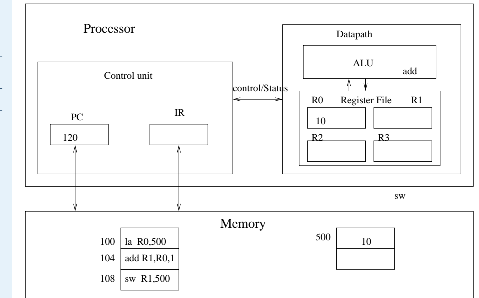
- Write Back de l'instruction load
- Accès mémoire de rien
- Exécution de l'instruction add (bypass)
- Décodage de l'instruction store



cycle 6

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

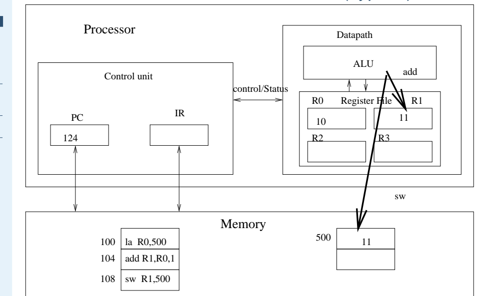
- Write Back de rien
- Accès mémoire de l'instruction add (rien)
- Exécution de l'instruction store (rien)



cycle 7

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Write Back de l'instruction add
- Accès mémoire de l'instruction store (bypass)



Bilan architecture non pipelinée

- Exécution non pipelinée:
  - 5 cycles pour exécuter une instruction
  - ⇒ 15 cycles pour 3 instructions.
- Exécution pipelinée:
  - 5 cycles pour exécuter une instruction
  - 8 cycles pour 3 instructions.
  - ⇒ sans branchement, une instruction par cycle
  - Un branchement (conditionnel ou pas) interrompt le pipeline car il faut attendre de décoder l'adresse de branchement pour charger l'instruction suivante ⇒ quelques cycles d'inactivité (pipeline stall)
  - Lors d'un branchement, certain ISA autorisent l'utilisation de ces *delai slots*: une ou deux instructions après le branchement sont exécutées, que le branchement soit pris ou pas (comme si elles étaint écrites avant le branchement).

introduction au MSP 430
Processeurs embarqués
Processeurs embarqués
Économie
ISA
Pipeline
Classification
Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Parallélisme au sein du processeur

Indépendamment du pipeline, Deux paradigmes dominants:

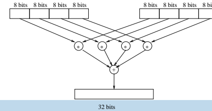
- Super Scalaire
  - Duplication des unités,
  - Répartition au vol des instructions sur les unités disponibles (re-ordonnancement des instructions: *out of order execution*)
  - Exemple: le PowerPC 970 (4 ALU, 2 FPU)
  - Efficace mais complexifie l'unité de contrôle (problème des interruptions)
- Very Large Instruction Word (VLIW)
  - Duplication des unités,
  - L'ordonnancement des instructions est fixé à la compilation (tout se passe comme si les instructions pouvait être regroupe sur 64 bits, 128 bits etc.)
  - Inventé par Josh Fisher (Yale) à partir du trace scheduling
  - Les processeurs VLIW sont tous basés sur les architecures RISC, avec entre 4 et 8 unités.
  - Exemple: TriMedia (Philips), Itanium IA64 (Intel).

introduction au MSP 430
Processeurs embarqués
Processeurs embarqués
Économie
ISA
Pipeline
Classification
Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Parallélisme au sein du processeur

Une autre approche possible: instructions SIMD.

- Modification du data-path pour proposer des opérations parallèles sur 16 ou 8 bits
- Exemple: Sun Visual Instruction Set, Intel Pentium MMX, Philips TriMedia
- Gains importants sur certains traitements mais très peu utilisé en pratique (difficile à inférer par le compilateur)
  - Librairies écrites en assembleur (programmes non portables)
  - Fonction C représentant les instructions assembleurs (*compiler intrinsic*)
  - Exemple: instruction `ifir8ii R1, R2, R3` du Trimedia:



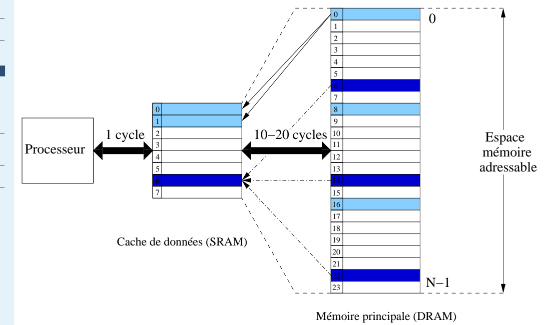
introduction au MSP 430
Processeurs embarqués
Processeurs embarqués
Économie
ISA
Pipeline
Classification
Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Mémoire

- Plusieurs technologies pour les mémoires:
  - Mémoires statiques (SRAM): petites, rapides, consommatrices, peu denses (chères).
  - Mémoires dynamiques (DRAM): grandes, lentes, très denses, transactions chères
- De plus en plus de place On-Chip pour la mémoire (dans ce cas elles sont moins efficaces que les chips mémoire).
- Ne pas oublier que le code aussi réside en mémoire
- Tous les systèmes ont des caches pour cacher les temps de latence lors de l'accès à la mémoire, en général plusieurs niveaux de caches: hiérarchie mémoire.

introduction au MSP 430
Processeurs embarqués
Processeurs embarqués
Économie
ISA
Pipeline
Classification
Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

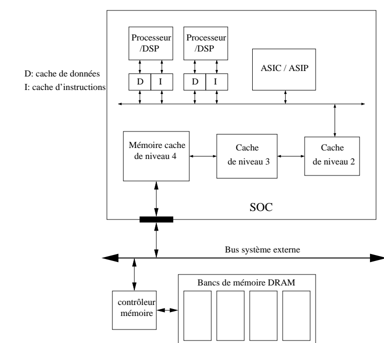
Principe du Cache



introduction au MSP 430
Processeurs embarqués
Processeurs embarqués
Économie
ISA
Pipeline
Classification
Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Hiérarchie Mémoire

D: cache de données
I: cache d'instructions



Différents types de processeurs embarqués

- Beaucoup de Processeurs à usage général ayant une ou deux générations
- 4, 8, 16 ou 32 bits (taille des mots)
- RISC et CISC
- DSP: Digital Signal Processor
- ASIP: Application Specific Integrated Processor

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

68000, x86

- Famille des Motorola 68000
  - Un des plus vieux processeur embarqué (ex Sun, Mac)
  - Architecture CISC
  - ISA propre et les meilleurs outils de développement, beaucoup d'utilisateurs
- Famille des x86
  - Démarre au 8086 (Intel) puis 80286, 386, 486, Pentium, et Athlon (AMD)
  - En processeurs embarqués: 5 fois moins que MIPS, ARM ou 68000.
  - architecture CISC, compatible avec le code du 8086
  - compatibilité mais mauvaises performances

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

SPARC, 29000 et i960

- SPARC
  - Un des premier RISC à avoir été embarqué (pratiquement plus aujourd'hui)
  - SPARC est une architecture brevetée (soft core, Intellectuel Property: IP), plusieurs compagnies fabriquent des SPARC
- 29000 (AMD)
  - Le 29000 a eu beaucoup de succès (imprimante laser Apple) grâce à ces 192 registres
  - AMD a arrêté la production car le développement des outils coûtait trop cher.
- i960 (intel)
  - Le i960 a été le plus vendu des processeurs embarqués au milieu des années 90 (router réseau et HP Laserjet).

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

MIPS, ARM, SuperH et PowerPC

- MIPS (microprocessor without interlocked pipeline stages)
  - Originellement pour les stations puissantes (SGI)
  - Puis, marché des consoles de jeux (Nitendo N64)
  - Famille très étendue: du plus gros (MIPS 20Kc, 64 bit) au plus petit (SmartMIPS, 32 bit pour carte à puce)
- ARM (Advanced RISC Machines, ex Acorn)
  - Un des 32 bits embarqués les plus populaires : téléphones portables
  - Faible consommation
  - Le successeur: StrongArm est commercialisé par Intel sous le nom de XScale
- SuperH (ou SH: Hitachi) Utilisé dans les stations Sega et les PDA
- PowerPC autant utilisé en embarqué qu'en ordinateur

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Et les autres....

- Plus de 100 processeurs embarqués 32 bits sur le marché
- Les constructeurs de FPGA proposent des soft-processeurs pour configurer les FPGA: Nios (Altera), MicroBlaze (Xilinx)
- Certain processeurs RISC (Crusoe de Transmetta) peuvent exécuter du code CISC (Intel)
  - Principe: recompilation du code à l'exécution (*runtime compilation*)
  - Gain obtenus par un mécanisme de cache, d'optimisation poussée des portion de code répétées (boucle), et grâce au parallélisme de niveau instruction
  - Réduction drastique de la consommation pour des performances équivalentes

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Micro-contrôleurs

- Utilisé pour le contrôle embarqué
  - Censeur, contrôleurs simples
  - Manipule des événements, quelques données mais en faible quantité
  - Exemple: camescope, disque dur, appareil photo numérique, machine à laver, four à micro-onde
- Quelques caractéristiques fréquentes
  - Périphériques présents sur le circuit (timer, convertisseur analogique numérique, interface de communication), accessible directement grâce aux registres
  - Programme et données intégrées au circuit
  - Accès direct du programmeur à de nombreuses broches du circuit
  - Instructions spécialisées pour les manipulation de bits.
- Le MSP430 appartient à cette catégorie

DSP: Digital Signal Processing

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Utilisés pour les applications de traitement du signal
  - Grande quantités de données numérisées, souvent organisées en flux
  - Filtre numérique sur téléphone, TV numérique, synthétiseur de sons
- Relativement proche des GPP, mais quelques caractéristiques en plus:
  - Bande passante élevée (deux bus)
  - Instructions dédiées pour les calculs de traitement du signal: multiplication accumulation,
  - Arithmétique spécifique (mode d'arrondi)
  - Registres dédiés pour certains opérateurs.
  - Constructeurs: Texas Instrument, puis Analog Devices, Motorola

Quelques mécanismes matériels utiles

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Densité de code:
  - La taille du code est importante pour les codes embarqués car elle influe sur la taille de la mémoire utilisée
  - Un programme C compilé pour SPARC prendra deux fois plus de place en mémoire que le même programme compilé pour le 68030.
  - En général les code RISC sont deux fois moins dense que les codes CISC (ex: instruction `TBLS` du 68300: *table lookup and interpolate*)
  - Les options de compilation doivent être utilisées avec précaution.
  - Le code est quelquefois stocké compressé et decompressé au vol par du matériel spécifique.

Quelques mécanismes matériels utiles

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Manipulations au niveau bit:
  - Utilisé pour les algorithmes de cryptage mais surtout pour les pilotes de périphériques.
  - La plupart des périphériques indiquent leur état au processeur en mettant un certain bit à 1 dans un certain registre.
  - Un processeur standard doit rapatrier le mot de 32 bit, masquer et tester à 0
  - Les instructions `BIC`, `BIT` et `BIS` du MSP430 font des manipulation au niveau bit dans la mémoire

Quelques mécanismes matériels utiles

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Données non-alignés
  - De nombreux traitements manipulent des données de taille non-multiple de 32 (paquets TCP/IP, video streams, clés d'encryption, 20 bits, 56 bits)
  - Les processeurs RISC savent uniquement transférer des mots (32 bits) *alignés* (calés sur une adresse multiple de 32 bits).
  - La plupart des architectures CISC (68k, x86) peuvent faire des chargements non alignés

Quelques mécanismes matériels utiles

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Gestion spécifique du cache
  - Les caches améliorent les performances mais introduisent du non-déterminisme.
  - Les contraintes spécifiques des systèmes embarqués ont entraîné des mécanismes particuliers pour les cache
  - On peut vouloir bloquer le cache (cache locking): forcer certaines données ou instruction à se charger et rester dans le cache (on parle aussi de mémoire *scratch-pad memory* ou de *software controlled cache*).
  - La plupart des caches utilisent une politique de Write-Back: une donnée modifiée dans le cache n'est pas forcément immédiatement recopiée en memoire. Dans le cas de périphériques mappés en mémoire, il est indispensable de recopier immédiatement (politique *write-through*)

Quelques mots sur la consommation

introduction au MSP 430
Processeurs embarqués
- Processeurs embarqués
- Économie
- ISA
- Pipeline
- Classification
- Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Trois composantes de la consommation d'une porte logique (inverseur)
  - Consommation dynamique : $P_{dyn} = C.V_{CC}^2$
    (C capacité de la porte)
  - Consommation statique : $P_{static} = V_{CC}.I_{leak}$
    (V_{CC}: tension d'alimentation, I_{leak} intensité des courants de fuite)
  - Consommation de court-circuit $P_{cs} = K.\tau.(V_{CC} - 2V_{Th})^3$.
    (K:constante technologique ; V_{Th}:tension seuil ; τ:temps de montée descente du signal)
- Aujourd'hui (2004) $P_{dyn} \gg P_{static} \gg P_{cs}$
- Demain (2007) $P_{dyn} \approx P_{static} \gg P_{cs}$

Consommation d'un circuit CMOS

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Généralisation naïve en prenant en compte une activité moyenne α (nombre moyen de portes commutant)
  - Consommation dynamique : $P_{dyn} = C.V_{CC}^2.\alpha.f$
    (f: fréquence du circuit)
  - Consommation statique : $P_{static} = V_{CC}.I_{leak}.N.k_{design}$
    (N: nombre de portes, k_{design} constante dépendant du design)
- Cette modélisation est très imprécise pour un circuit dont le comportement n'est pas stationnaire (ex: processeur)

Réduction statique de la consommation

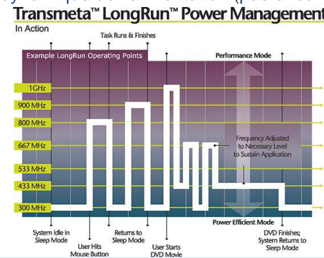
introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Le facteur le plus important est la tension d'alimentation (V_{CC}) d'abord 3.3 V puis 2.5 V. Les versions récentes de Xscale (strong ARM, Intel) et les puces smartCard fonctionnent a 0.65 V
- On peut différencier les tensions en fonction du bloc du chip: 1.5 V pour le processeur, 3.3 pour l'interface du bus et les pattes d'entrée/sortie (ex: Strong ARM de Digital)
- Plus la technologie est intégrée, moins elle consomme (capacité diminuée).
- Fréquence d'horloge peu élevée compensée par le parallélisme
- Complexité réduite des différents composants (moins de registres, architectures RISC)

Réduction dynamique de la consommation

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Gestion dynamique de la fréquence d'horloge
- Exemple: processeur Crusoe (Transmetta)
  Suppression de l'horloge sur un bloc (*Dynamic clock gating*)
- Gestion dynamique de l'alimentation (pas encore réalisé)



Low Power Mode pour le MSP430

introduction au MSP 430
Processeurs embarqués
● Processeurs embarqués
● Économie
● ISA
● Pipeline
● Classification
● Charactéristiques
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Différent mode pour réduire la consommation
  - LPM0: le CPU est arrété
  - LPM1, LPM2: l'horloge rapide (`MCLK`) est aussi arrétée
  - LPM3 le générateur d'horloge est arrété
  - LPM4 : l'oscillateur du crystal est arrété
- Le temps de reprise est d'autant plus long que la veille est profonde.

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
● Outils GNU pour la compilation
● GNU
● Compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Outils GNU pour la compilation

Le projet GNU

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
● Outils GNU pour la compilation
● GNU
● Compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

- Le projet GNU, initié par Richard Stallman en 1984, vise à créer un système d'exploitation complet qui réponde aux principes du logiciel libre
- Licence GPL (GNU General Public License) ou LGPL (GNU Lesser General Public License)
- Philosophie unix: programmes modulaires assemblés en un système complexe:
  - Portabilité
  - Standard Posix
  - Performances
- Prise en main plus difficile que les outils commerciaux.
- Modèle économique de plus en plus suivi par les industriels.

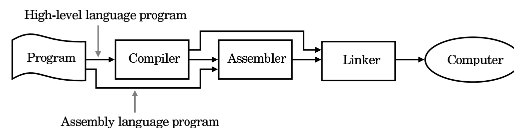
principaux outils de développement GNU

- Le compilateur `gcc` est la pièce maîtresse du projet GNU. Compilateur C, C++ et Objective-C de très grande qualité, ses performances dépassent souvent celles des meilleurs compilateurs commerciaux.
- Autres compilateurs `g77`, `gnat`, `gpc`,...
- `emacs` éditeur de texte multifonction qui peut faire aussi office d'environnement intégré de programmation (IDE)
- `make` permet d'automatiser l'ordonnancement des différentes étapes de compilation
- `gdb` est un débogueur pour les langages C, C++ et Fortran. `ddd` apporte une interface graphique au-dessus de gdb.
- `automake, autoconf` permette de produire facilement des programmes portables.
- Tous ces programmes sont disponibles sur tous les type de systèmes. Pour windows, c'est à travers l'environnement `cygwin`

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
- Outils GNU pour la compilation
- GNU
- Compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Compilation: Le flot général

- Le flot complet de compilation est le suivant:

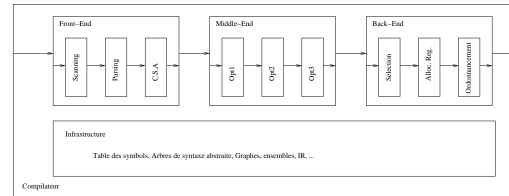


- La programmation d'un système embarqué nécessite souvent d'écrire explicitement des parties en assembleur (pilotes de périphériques et d'accélérateurs matériels).

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
- Outils GNU pour la compilation
- GNU
- Compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Détail de la partie “compilation”

- le processus de compilation est divisé en trois phases:



introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
- Outils GNU pour la compilation
- GNU
- Compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Compilation: Le front-end

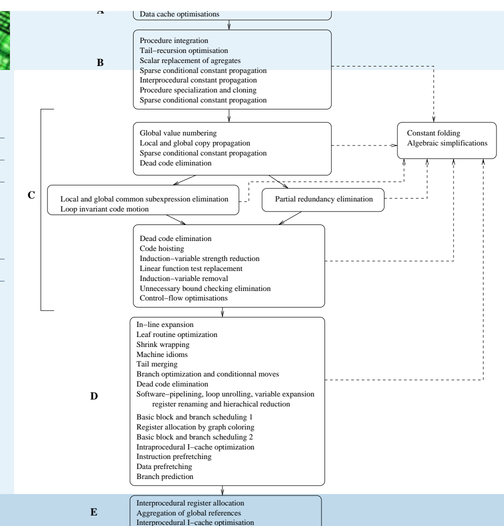
- Le front-end d'un compilateur pour code embarqué utilise les mêmes techniques que les compilateurs traditionnels (on peut vouloir inclure des partie d'assembleur directement)
- Parsing LR(1): Le parseur est généré à partir de la grammaire du langage.
- Flex et bison: outils GNU

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
- Outils GNU pour la compilation
- GNU
- Compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution

Compilation: Le middle-end

- Certaines phases d'optimisations sont ajoutées, elles peuvent être très calculatoires
- Quelques exemples de transformation indépendantes de la machine:
  - Élimination d'expressions redondantes
  - Élimination de code mort
  - Propagation de constantes
- Attention, les optimisation peuvent nuire à la compréhension de l'assembleur (utiliser l'options -O0 avec `gcc`)

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
- Outils GNU pour la compilation
- GNU
- Compilation
Un compilateur pour l'embarqué: GCC
Pile d'exécution



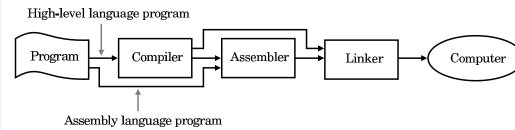
Compilation: Le back-end

- La phase de génération de code est dédiée à l'architecture cible. Les techniques de compilation reciblable sont utilisées pour des familles d'architectures.
- Les étapes les plus importantes sont
  - Sélection de code
  - Allocation de registre
  - Ordonnancement d'instructions

Un compilateur pour l'embarqué: GCC

- GCC: Gnu C Compiler ou Gnu Compiler Collection
- http://gcc.gnu.org/
- Outil développé par la communauté mondiale, développement rapide.
- De plus en plus utilisé pour le calcul embarqué car il est reciblable.
- Exemple d'utilisation
  - Créer un compilateur pour le MSP430 sur votre Pentium
  - Insérer une routine d'interruption dans votre programme
  - Répartir les différentes sections de votre code dans différentes mémoires.

compilateur, éditeur de liens, binutils



- gcc: `ftp://ftp.gnu.org/gnu/gcc/gcc-3.3.tar.gz`
- Assembleur, éditeur de lien et utilitaire de manipulation de binaire (`objdump`, etc..) `ftp://ftp.gnu.org/gnu/binutils/binutils-2.9.1.tar.gz`
- Run-time library (`printf`, `malloc`, etc.): `newlib` (ou `glibc`) `ftp://ftp.gnu.org/gnu/glibc/`

Installation de GCC pour MSP430

- `tar zxf gcc-3.3.tar.gz`
- `tar zxf binutils-2.9.1.tar.gz`
- `cd binutils-2.9.1`
  ```
  configure
   -target=msp430
  make all install
  ```
- Même chose pour `gcc` (L'exécutable est nommé `msp430-gcc`)
- Voir le fichier configure.sub pour les plate-formes cibles possibles
- On peut aussi compiler gcc sur une machine pour l'utiliser sur une autre machine (ou il produira du code pour une troisième machine): GCC est un *Cross-compiler*

GCC

- La commande `gcc` lance plusieurs programmes suivant les options
  - Le pré-processeur `cpp`
  - Le compilateur `cc1`
  - L'assembleur `gas`
  - L'éditeur de liens `ld`
- `gcc -v` permet de visualiser les différents programmes appelés par `gcc`

Le pré-processeur: `cpp` ou `gcc -E`

- `cpp0` pour msp430-gcc
- `msp430-gcc -mmcu=msp430x449 -E ex1.c -o ex1.i`
- Les tâches du préprocesseur sont :
  - élimination des commentaires,
  - l'inclusion de fichier source,
  - la substitution de texte,
  - la définition de macros,
  - la compilation conditionnelle.
- Exemple:

ex1.c
```
#define MAX(a, b) ((a) > (b) ? (a) : (b))
...
f=MAX(3,b);
```

ex1.i
```
#define MAX(a, b) ((a) > (b) ? (a) : (b))
...
f=((3) > (b) ? (3) : (b));
```

Le compilateur cc1 ou gcc -S

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
- GCC
- gcc
- gdb
- Pile d'exécution
Pile d'exécution

- Génère du code assembleur
- `msp430-gcc -O0 -mmcu=msp430x449 -S main.c -o main.S`
- Exemple :

```
void main()
{ int i;
  i=0;

  while (1)
   {
     i++;
     nop();
   }
}
```


Assembleur produit par mspgcc -S

```
        .text
        .p2align 1,0
.global         main
        .type         main,@function
main:
/* prologue: frame size = 2 */
.L__FrameSize_main=0x2
.L__FrameOffset_main=0x6
        mov       #(__stack-2), r1
        mov       r1,r4
/* prologue end (size=3) */
        mov       #llo(0), @r4
.L2:
        add       #llo(1), @r4
        nop
        jmp       .L2
/* epilogue: frame size=2 */
        add       #2, r1
        br        #__stop_progExec__
/* epilogue end (size=3) */
```


Éditeur de liens: ld

- Produit l'exécutable (a.out par défaut) à partir des codes objets des programmes et des librairies utilisées.
- Il y a deux manières d'utiliser les librairies dans un programme
  - Librairies dynamiques ou partagées (*shared*, option par défaut): le code de la librairie n'est pas inclus dans l'exécutable, le système charge dynamiquement le code de la librairie en mémoire lors de l'appel du programme. On n'a besoin que d'*une* version de la librairie en mémoire même si plusieurs programmes utilisent la même librairie. La librairie doit donc être *installée* sur la machine, avant d'exécuter le code.
  - Librairies statiques (*static*): le code de la librairie est inclus dans l'exécutable. Le fichier exécutable est plus gros mais on peut l'exécuter sur une machine sur laquelle la librairie n'est pas installée.
- Les OS simples ne proposent pas de librairies dynamiques.

Assembleur correspondant (vu par gdb-insight)

```
 while (1)
   { i++;
     nop();
   }
```

```
mov     #2558,  SP          ; initialisation de la pile
mov     r1,  r4             ; r4 contient  SP
mov     #0,  0(r4)          ; initialisation de i
inc     0(r4)               ; i++
nop                         ; nop();
jmp     $-6                 ; saut inconditionnel (PC-6):
incd     SP                 ;
br     #0x1158               ;
```


Assembleur as ou gas

- Transforme un fichier assembleur en un code objet (représentation binaire du code assembleur)
- L'options -c de gcc permet de combiner compilation et assemblage:
  `msp430-gcc -c -mmcu=msp430x449 main.c -o main.o`

Manipuler les fichiers binaires

Quelques commandes utiles:

- nm
  permet de connaître les symboles (en particulier les fonctions) utilisés dans un fichier objet ou exécutable:

```
trisset@hom\$ msp430-nm fib.elf | grep main
000040c8 T main
```

- objdump permet d'analyser un fichier binaire. Par exemple pour avoir la correspondance entre la représentation binaire et le code assembleur:

```
trisset@hom$ msp430-objdump -f fib.elf
fib.elf:      file format elf32-msp430
architecture: msp:43, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00001100
```

Options utiles de gcc

- `-c` :pas d'édition de liens
- `-o file`: renomme le fichier de sortie `file` au lieu de `a.out`
- `-g` : insère les informations nécessaires à l'utilisation d'un débogueur (`gbd, ddd`).
- `-Wall` : fait le maximum de vérifications statiques possibles
- `-Oval` :(`val` est un entier compris entre 0 et 4), effectue des optimisations de niveau `val`
- `-Ipath` : recherche les fichiers d'en-tête dans le répertoire `path` avant de les rechercher dans les répertoires standards (`/usr/include`, `/usr/local/include`).
- `-Lpath` : recherche les librairies dans le répertoire `path` avant de les rechercher dans les répertoires standards (`/usr/lib`, `/usr/local/lib`).
- `-Dflag=val` : équivalent à écrire la directive `#define flag val` dans le code

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Attention aux optimisation: -01

- Assembleur du main (`while(1) {i++; nop()}`) avec l'option -O1 `msp430-gcc -O1 -mmcu=msp430x449 -S main.c -o main.S`

```
while (1)
  { i++;
    nop();
  }
```

```
mov    #2558,  SP
nop
jmp    $2
br    #0x114C
```


introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

gdb: GNU symbolic debugger

- `gdb` est un debugger symbolique, c'est-à-dire un utilitaire Unix permettant de contrôler le déroulement de programmes C.
- `gdb` permet (entre autres) de mettre des points d'arrêt dans un programme, de visualiser l'état de ses variables, de calculer des expressions, d'appeler interactivement des fonctions, etc.
- `xxgdb`, `ddd` et `insight` sont des interfaces graphiques qui facilitent l'utilisation de `gdb` sous X-Window.
- `gdb` ne nécessite aucun système de fenêtrage, il peut s'exécuter sur un simple terminal shell (mode *console*).
- Il est indispensable de comprendre le fonctionnement en mode de `gdb` pour pouvoir utiliser `insight`.

Exemple de session gdb

```
#include <stdio.h>

int main()
{int i,*p;

  i=1;
  p=(int *)malloc(sizeof(int));
  *p=2;
  *p+=i;
  fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
  free(p);
  return(0);
}
```

- compilation avec `-g`:
  `shell$ gcc -g exgdb.c -o exgdb`
- lancement de `gdb`: `shell$ gdb exgdb`
  `GNU gdb 6.3-debian`
  `(gbd)`

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Exemple de session gdb

- Lorsque l'on lance `gdb` avec la commande `gdb exgdb.c`, `gdb` a chargé l'exécutable, il attend alors une commande `gdb`, comme par exemple `run` (pour exécuter le programme), `break` (pour mettre un point d'arret dans le programme), `step` (pour avancer d'une instruction dans le programme), etc.
- Les points d'arrêt peuvent se positionner au début des fonctions (`break main`, par exemple), ou à une certaine ligne (`break 6`, par exemple), ou lorsqu'une variable change de valeur (`watch i`, par exemple).

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Exemple de session gdb

- Suite de la session:

```
(gdb) break main
Breakpoint 1 at 0x8048424: file exgdb.c, line 6.
(gdb) run
Starting program: /home/trisset/cours/2005/AGP/cours_t

Breakpoint 1, main () at exgdb.c:6
6         i=1;
(gdb)
```

`gdb` a lancé l'exécutable et arrêté l'exécution à la ligne 6 du fichier (à la première ligne de la fonction `main`), le code de cette ligne apparaît à l'écran.

- On peut avancer d'un pas dans le programme:

```
(gdb) step
7         p=(int *)malloc(sizeof(int));
(gdb)
```

- On peut afficher la valeur de i;

```
(gdb) print i
$1 = 1
(gdb)
```

Exemple de session gdb

- Affichage permanent de variables: `display`

```
(gdb) display i
1: i = 1
(gdb)
```

- Affichage de pointeurs (comme une variable):

```
(gdb) display p
2: p = (int *) 0xb8000540
(gdb)
```

- Affichage d'objet pointés par les pointeurs :

```
(gdb) display *p
3: *p = -1208053760
(gdb)
```

```
(gdb) step
8         *p=2;
3: *p = 0
2: p = (int *) 0x804a008
1: i = 1
(gdb)
```


introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Exemple de session gdb

```
(gdb) step
9         *p+=i;
3: *p = 2
2: p = (int *) 0x804a008
1: i = 1
```

```
(gdb) step
10        fprintf(stdout,"i=%d, p=%X, *p=%d\n",i,p,*p);
3: *p = 3
2: p = (int *) 0x804a008
1: i = 1
```

```
(gdb) next
i=1, p=804A008, *p=3
11        free(p);
3: *p = 3
2: p = (int *) 0x804a008
1: i = 1
```

```
(gdb) cont
Continuing.

Program exited normally.
(gdb)
```


introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

gdb commandes abrégées

- On peut taper la première lettres des commandes:
- `r` est équivalent à `run`
- `b main` est équivalent à `break main`
- `p var` est équivalent à `print var`
- `d var` est équivalent à `display var`
- `s` est équivalent à `step`
- `n` est équivalent à `next`
- `c` est équivalent à `continue`
- La commande `run` peut prendre des arguments, le programme est alors exécuté avec les arguments donnés
- la commande `info` permet d'afficher des informations sur l'état du programme dans le débugger. Par exemple `info b` liste les points d'arrêt.
- La commande `help` est l'aide en ligne de `gdb`

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

insight

- `insight` est une interface graphique pour `gdb`.
- Après avoir compilé un programme avec `msp430-gcc -g`, on tape:
  `msp430-gdb-insight nomduprog`
  Par exemple:
  `msp430-gdb-insight fib.elf`
- Le débuggage peut se faire dans `insight` (mode simulateur) ou avec une plateforme externe (connection TCP).

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

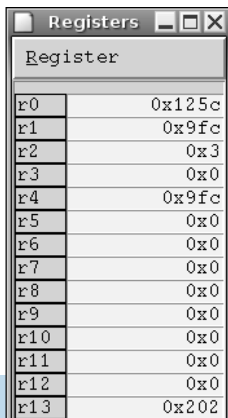
insight: haut de la fenêtre

```
main.c - Source Window
File  Run  View  Control  Preferences
0x125c
main.c    main    SOURCE
   80 interrupt (PORT1_VECTOR) prvbutton( void ); // __attr
   81 interrupt (PORT1_VECTOR) prvbutton( void )
-  82 {
-  83   int val = P1IN;
-  84   P4OUT = val;
-  85 }
   86
   87 /**
   88 Main function with some blinking leds
   89 */
-  90 int main(void) {
   91     int i;
   92     int val;
   93
   94     //WDTCTL = WDTCTL_INIT;
   95
■  96     P1OUT  = P1OUT_INIT;  //Init output data of port1
-  97     P2OUT  = 0;
-  98     P3OUT  = 0;
-  99     P4OUT  = 0;
  100
```

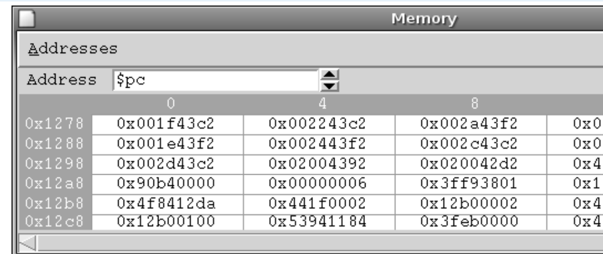
introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

```
   87 /**
   88 Main function with some blinking leds
   89 */
-  90 int main(void) {
   91     int i;
   92     int val;
   93
   94     //WDTCTL = WDTCTL_INIT;
   95
■  96     P1OUT  = P1OUT_INIT;  //Init output data of port1
-  97     P2OUT  = 0;
-  98     P3OUT  = 0;
-  99     P4OUT  = 0;
  100
- 101     P1SEL  = P1SEL_INIT;  //Select port or module -fu

-    0x1256 <main>:        mov      #2556,  SP
-    0x125a <main+4>:      mov      r1,     r4
■    0x125c <main+6>:      mov.b    #0,     &0
-    0x1260 <main+10>:     mov.b    #0,     &0
-    0x1264 <main+14>:     mov.b    #0,     &0
-    0x1268 <main+18>:     mov.b    #0,     &0
-    0x126c <main+22>:     mov.b    #0,     &0
-    0x1270 <main+26>:     mov.b    #0,     &0
-    0x1274 <main+30>:     mov.b    #0,     &0
-    0x1278 <main+34>:     mov.b    #0,     &0
-    0x127c <main+38>:     mov.b    #0,     &0
-    0x1280 <main+42>:     mov.b    #-1,    &0
-    0x1284 <main+46>:     mov.b    #-1,    &0
-    0x1288 <main+50>:     mov.b    #-1,    &0
-    0x128c <main+54>:     mov.b    #-1,    &0

Program stopped at line 97, 0x1260
```

insight: fenètre des registres du MSP



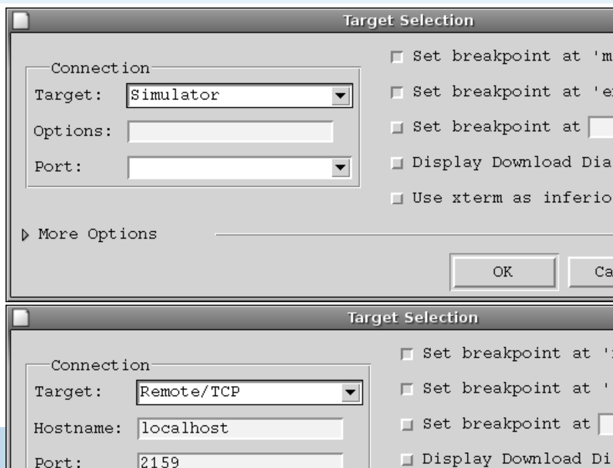
insight: état de la mémoire du MSP



Outils de GCC

Utilisation de GCC pour le développement de code embarqué

insight: connexion au programme



Liens à connaître

- make, documentation
  http://www.gnu.org/software/make/manual/make.html
- gcc home page: http://gcc.gnu.org/
- gdb documentation
  http://www.gnu.org/software/gdb/documentation/,

Assembleur dans le code C

- On peut include directement des instructions assembleur dans le code C avec la fonction `__asm__`

```
int main(void) {
    int i;
    int *p,*res;

    __asm__("mov #304, R4");     // p=0x130;
    __asm__("mov #2, @R4");      // *p=2;
    __asm__("mov #312, R4");     //p=0x138;
    __asm__("mov #5, @R4");      //*p=5;
    __asm__("mov #314, R4");
    __asm__("mov @R4, R5");      //Res=mem(0x13A);

    nop();
}
```

- Permet d'écrire des pilotes de périphériques, de contrôler la gestion des interruptions sans système d'expoitation

Assembleur dans le code C

- On peut aussi mettre explicitement des variables dans des registres sans connaître l'allocation de registres faite par le processeur
- Exemple: utilisation de la fonction `fsinx` du 68881:
  `__asm__("fsinx %1,%0" : "=f" (result) : "f" (angle));`
- %0 et %1 représentent le résultat et l'opérande de la fonction qui vont correspondre aux variables `result` et `angle` du programme C
- `"f"` est une directive indiquant à `gcc` qu'il doit utiliser des registres flottants

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Contrôler les section du programme

- Le code assembleur contient différentes sections. Par exemple avec GCC
  - La section `.text` contient les instructions du programme
  - La section `.data` contient des données statiques etc.
- Le concepteur de logiciel embarqué veut souvent contrôler explicitement la répartition des variables globales dans les sections (à la compilation): pour distinguer les variables des constantes par exemple.
- Il peut vouloir aussi contrôler la répartition des sections dans les composants matériels (à l'édition de lien et au chargement du programme)
- Utilisation de la directive `__attibute__`

```
const int put_this_in_rom
      __attribute__((section("myconst" )));
const int put_this_in_flash
      __attribute__((section("myflash" )));
```


introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Gestionnaire d'interruption

- Beaucoup de périphériques dédiés communiquent avec le processeur par des interruptions
  - Problèmes (adresse mémoire invalide), erreur de transmission sur le bus
  - Fin de tache pour un accélérateur matériel
  - Détection de données pour un senseurs
- GCC ne peut pas généralement pas directement compiler un gestionnaire d'interruption (retour par `rte`: return from exception)
- on peut contourner ce problème en encapsulant la procédure de gestion de l'interruption

```
/* code C de gestion de l'interruption */
 void isr_C( void ) {
  /* ISR: interupt service routine
    faire quelque chose en C */
  ...
}

/* code assembleur utilisé
   dans le code source */
 __asm__("
  .global _isr
  _isr:
  /* Sauvegarde des registres
   * choisis
   */
  push r0
  push r1
  ...
  /* appel du gestionnaire */
  jsr _isr_C
  /* restauration des registre
   * et retour
   */
  ...
  pop r1
  pop r0
  rte
");
```


introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Gestionnaire d'interruption pour MSP430

- Le compilateur `msp430-gcc` dispose de facilité pour cela
- Le mot cles `interrupt (CODE_INTERRUPT)` permet de déclarer une fonction comme handler d'interruption et de faire les traitement supplémentaires par rapport à un appel de fonction normal (ici essentiellement sauvegarde du `SP` sur la pile).
- Le compilateur dispose aussi d'un certain nombre de macro pour désigner les ports, les diverses paramètres du MSP, elles sont dans le répertoire `/include/msp430`, là ou les outils `mspgcc` ont étés installé. En particulier
  - `iomacro.h`
  - `msp430x44x.h`
  - `msp430/basic_timer.h`
  - ...

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Le LD script

- L'éditeur de liens assemble les différents fichiers objets résultant de la compilation séparée des différents fichiers.
- C'est là que sont résolus les appels à des fonctions entre fichiers ou à des fonctions de bibliothèques non fournies par l'utilisateur
- C'est aussi là que sont agencées les différentes sections mémoire dans l'espace d'adressage final.

introduction au MSP 430
Processeurs embarqués
Outils GNU pour la compilation
Un compilateur pour l'embarqué: GCC
● GCC
● gcc
● gdb
● Pile d'exécution
Pile d'exécution

Exemple de LD script (MIPS)

- Spécification de fichier de librairie
- Spécification du format de sortie
- 5 sections nommées

```
/* Une list de fichier à inclure (les autres sont
     spécifiés par la ligne de commande */
INPUT(libc.a libg.a libgcc.a libc.a libgcc.a)

/* Specification du format de sortie
(binaire  :bin}, Intel Hex:
ihex, debug coff-\$target */
  OUTPUT_FORMAT("coff-sh")


/* list of our memory sections */
MEMORY {
 vect : ORIGIN = 0x00000000, LENGTH = 1k
 rom  : ORIGIN  = 0x00000400, LENGTH = 127k
 reset: ORIGIN = 0xBFC00000, LENGTH = 0x00000400
 ram  : ORIGIN = 0x400000, LENGTH = 128k
 cache : ORIGIN = 0xfffff000, LENGTH = 4k
}
```

Exemple de LD script (suite)

- Description du placement de chaque section en mémoire
- Création d'un symbole au début de la section (ex: `__text_start`) et à la fin (`__text_end`)
- Placement des parties du code préfixé par la directive `.text` dans la section `rom` de la mémoire.
- Éventuellement insertion de code spécifiquement écrit directement dans la mémoire (`.reset`)

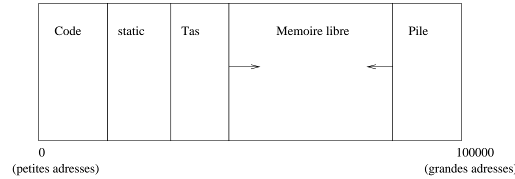
```
SECTIONS {
 /* the interrupt
    vector table */
 .vect :
 {
  __vect_start = .;
  *(.vect);
  __vect_end = .;
 } > vect

 /* code and constants */
 .text :
 {
  __text_start = .;
  *(.text)
  *(.strings)
   __text_end = .;
 } > rom

.reset : {
   ./libhandler.a(.reset)
   } > reset
```

```
/* uninitialized data */
.bss :

{
  __bss_start = . ;
 *(.bss)
 *(COMMON)
  __bss_end = . ;
} > ram

/* initialized data */
.init : AT (__text_end)
{
 __data_start = .;
 *(.data)
 __data_end = .;
} > ram

/* application stack */
.stack :
{
  __stack_start = .;
 *(.stack)
  __stack_end = .;
} > ram
}
```


Pile d'exécution

Pile d'exécution

- Le mécanisme de transfert de contrôle entre les procédures est implémenté grâce à la *pile d'exécution.*
- Le programmeur à cette vision de la mémoire virtuelle:



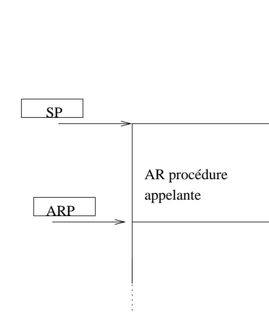
- Le tas (*heap*) est utilisé pour l'allocation dynamique.
- La pile (*stack*) est utilisée pour la gestion des contextes des procédures (variable locales, etc.)

Enregistrement d'activation

- Appel d'une procédure: empilement de l'*enregistrement d'activation* (AR pour *activation record*).
- L'AR permet de mettre en place le *contexte* de la procédure.
- Cet AR contient
  - L'espace pour les variables locales déclarées dans la procédure
  - Des informations pour la restauration du contexte de la procédure appelante:
    - Pointeur sur l'AR de la procédure appelante (ARP ou FP pour *frame pointeur*).
    - Adresse de l'instruction de retour (instruction suivant l'appel de la procédure appelante).
    - Éventuellement sauvegarde de l'état des registres au moment de l'appel.

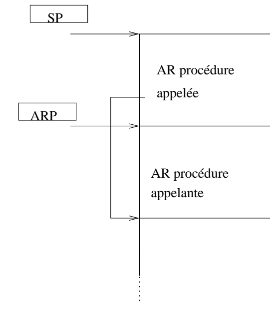
Appel de procédure: état de la pile

avant l'appel

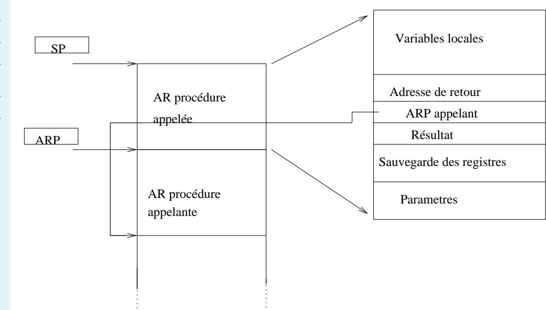


⇒

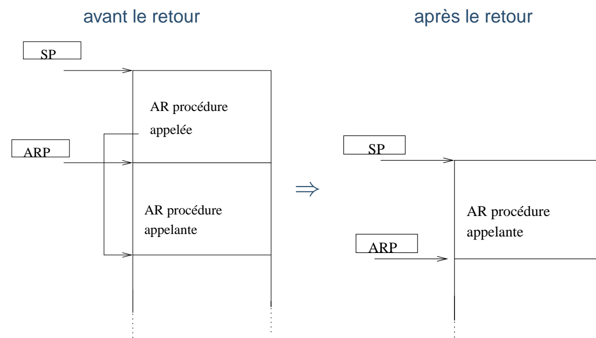
après l'appel



Contenu de l'AR



Retour de procédure: état de la pile



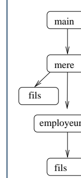
Lien statique et Lien dynamique

- Considérons une procédure `employeur` qui appelle une procédure `fils`.
- Dans l'AR de `fils`, l'ARP appelant pointe sur l'AR de `employeur`.
- Ce pointeur l'ARP est quelquefois appelé le *lien dynamique*, il pointe sur l'environnement de la procédure appelante (ici `employeur`).
- Considérons maintenant la procédure `mère` dans laquelle `fils` à été déclarée.
- Dans certains langages comme Pascal, la procédure `fils` peut accéder aux variables de `mère`
- Pour cela on a besoin d'un *lien statique* qui est un pointeur sur l'environnement de la procédure ou l'on a été déclaré.

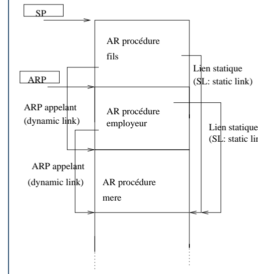
Exemple: arbre d'appel

procedure $main()$;
  var y...
  procedure $mere()$
    var z: ...
    procedure $fils()$;
      begin { $fils()$}
        ...
      end; { $fils()$}
    procedure $employeur()$;
      var y....
      begin { $employeur()$}
        x:=...
        $fils()$;
      end;
    begin { $mere()$}
      z:=1;
      $fils()$;
      $employeur()$;
    end;
  begin { $main()$}
    $mere()$;
    ...
  end;

Arbre d'appel:



Pile (lors du 2eme appel de fils



Exemple: fibonnacci sur MSP

```
int fib (int i)
{
  if (i<=1) return(1);
  else return(fib(i-1)+fib(i-2));
}
```


Code assembleur fibonnacci (-O0)

```
push     r11           ;sauvegarde des registres caller-save r11 sur la pile
push     r5            ;    (cloberred): R4-R11
push     r4            ;
mov     r1, r5         ;
add     #8, r5         ;R5 pointe dans la pile
sub     #4, SP         ;reserve deux nouveaux entiers dans la pile
mov     r1,  r4        ;R4 pointe vers le somme de pile
mov     r15, 0(r4)     ;mise en place de l'argument i en var. locale
cmp     #2, 0(r4)      ;comparaison de i à 2
jge     $+8            ;branchement si i>=2 (instr mov @R4, R5)
mov     #1, 2(r4)      ;sinon mise en place du resultat (SP+2)
jmp     $+26           ;saut à    mov      2(r4),      r15
mov     @r4, r15       ;Si i>2 R15 <- i
add     #-1, r15       ;R15 <- i-1
call     #4826         ;call fib
mov     r15, r11       ;resultat dans r15 -> R11 <- fib(i-1)
mov     @r4, r15       ;R15 <- i
decd     r15           ;R15 <- i-2
call     #4826         ;call fib
add     r15, r11       ;R11 <- fib(i-1) + fib(i-2)
mov     r11, 2(r4)     ;mise en place du resultat dans (SP+2)
mov     2(r4), r15     ;mise en place du resultat dans R15
add     #4, SP         ;on libère les deux entier
pop     r4             ;restauration R4
pop     r5             ;restauration R5
pop     r11            ;restauration r11
ret                    ;retour au code appelant
```


Code assembleur fibonnacci (options -O2)

```
0x1258    <fib>:       push   r11          ;
0x125a    <fib+2>:     push   r10          ;
0x125c    <fib+4>:     mov    r15, r10     ;
0x125e    <fib+6>:     cmp    #2, r15      ;subst r3 with As==10
0x1260    <fib+8>:     jl     $+24         ;abs dst addr 0x1278
0x1262    <fib+10>:    add    #-1,  r10    ;subst r3 with As==11
0x1264    <fib+12>:    mov    r10,  r15    ;
0x1266    <fib+14>:    call   #4696        ;#0x1258
0x126a    <fib+18>:    mov    r15,  r11    ;
0x126c    <fib+20>:    add    #-1,  r10    ;subst r3 with As==11
0x126e    <fib+22>:    mov    r10,  r15    ;
0x1270    <fib+24>:    call   #4696        ;#0x1258
0x1274    <fib+28>:    add    r11, r15     ;
0x1276    <fib+30>:    jmp    $+4          ;abs dst addr 0x127a
0x1278    <fib+32>:    mov    #1, r15      ;subst r3 with As==01
0x127a    <fib+34>:    pop    r10          ;
0x127c    <fib+36>:    pop    r11          ;
0x127e    <fib+38>:    ret
```