Performance
&
Quelques processeurs réels

IFT1225

Dép. d'IRO, Université de Montréal
(professeur E. Cerny)

Sujets

- Mesure de performance et caractéristiques des processeurs
- CISC versus RISC
- Exemple de CISC: Motorola série 68000
- Exemple de RISC: SUN SPARC

Efficacité vs. coût

- Le coût du travail utile est la question fondamentale
- Coût du chassis, planche de circuit imprimé, clavier, dominent le prix du circuit intégré
- Compatibilité preserve l'investissement en logiciel
  - Binaire
  - Source
  - Emulation
- Performance: une fonction de l'application, pas de mesure fiable universelle
- Perspective d'achat: meilleure performance, min. coût, performance/coût?
- Perspective face aux choix de design: meilleure amélioration de performance, min. coût, performance / coût?
- Les deux exigent une base pour la comparaison, mesure pour évaluer

Performance

- Temps pour effectuer un tâche: temps d'exécution, temps de réponse, latence
  - plus petit est meilleur
  - Sur n programmes:
    - Moyenne arithmétique: $Temps = \sum_{i=1}^{n} Temps_i$
- Nombre de tâches par unité de temps: performance, débit, taux de transfert (bande passante)
  - plus grand est meilleur
  - appareil A est n fois plus rapide que B:
    n = performanceA / performanceB
    - Moyenne harmonique: $Taux = \frac{n}{\sum \frac{1}{Taux_i}}$
- Les deux types de mesures donnent des informations complémentaires, leur utilité dépend de l'application

Bases d'évaluation

- Tâches réelles
  - spécifique à la situation, n'est pas portable, difficile à exécuter et mesurer
- Programmes tests basés sur des applications réelles
  - portable, le plus souvent utilisé, moins spécifique
- Noyaux d'algorithmes ("Kernel Benchmarks") synthétiques
  - faciles à exécuter, faciles pour falsifier les résultats (mettre au point le compilateur pour ces programmes tests, mais en réalité…?)
- Microtests ("Microbenchmarks") synthétiques
  - combinaisons spécifiques d'instructions, boucles d'itération, etc.
  - utile lors de design pour identifier des problèmes
  - mesurer la performance maximale atteignable

Mesures de performance

- **MIPS**: *Millions of Instructions Per Second*
  - Sauf que le même travail peut prendre différents nombres d'instructions sur des processeurs différents
- **MFLOPS**: *Million Floating Point OPs Per Second*
  - Autres instructions comptées comme temps système; faut avoir un jeu d'instructions à V.F. uniforme pour fins de comparaison
- **Whetstones**: Programmes de mesure (*benchmark*) synthétique
  - Un programme composé pour vérifier des traits spécifique de performance
- **Dhrystones**: Compétiteur synthétique de Whetstone
  - Construit pour corriger l'emphase de Whetstone sur virgule flottante
- **SPEC**: Collection de programmes réels du monde C et Unix de différents domaines

SPEC95

- 18 programmes tests (benchmarks) avec des données
- Applications plutôt en sciences / techniques
- 8 traitement en nombres entiers
  - go, m88ksim, gcc, compress, li, ijpeg, perl, vortex
- 10 traitement en nombres à virgule flottante
  - tomcatv, swim, su2cor, hydro2d, mgrid, applu, turb3d, apsi, fppp, wave5
- Faut les exécuter avec les contrôle de compilation standard pour éliminer des optimisations qui ciblent ces programmes tests seulement

Mesures de performance

- Application: Réponses par unité de temps, opérations utiles / seconde
- Architecture du jeux d 'instructions: MIPS, MFLOPS
- Chemin de données, contrôle: Mégaoctets par seconde
- Unités fonctionnelles, contrôle, logique: Cycles d 'horloge par seconde

- Chaque mesure peut être mal utilisée…
  voir MIPS de RISC et de CISC….

- Composants de la performance d 'une unité centrale:

$$Temps\_CPU = \frac{secondes}{} = \frac{instructions}{} \times \underbrace{\frac{cycles}{i} \times \frac{secondes}{i}}_{\mathbf{CPI}}$$

Cycles per instruction

$$CPI = (Temps\ CPU \times Fréquence\ d'horloge) / Nombre\ d'instructions$$
$$= Nb\ de\ cycles / Nombre\ d'instructions$$
$$= \sum_{i=1}^{n} CPI_i \times F_i$$

où $F_i = \dfrac{Nombre\ d'instructions\ I_i}{Nombre\ d'instructions\ total}$

Nb d 'instructions exécutées, pas statiquement dans le programme!!

$$Temps\ CPU = Période\ d'horloge \times CPI \times Nombre\ d'instructions\ total$$
$$= Période\ d'horloge \times \sum CPI_i \times Nombre\ d'instructions\ I_i$$

CPI - exemple

Opération	Fréq F_i%	Nb de Cycles	CPI_ixF_i	% de temps
ALU	**50**	**1**	**0.5**	**23**
Load	**20**	**5**	**1.0**	**45**
Store	**10**	**3**	**0.3**	**14**
Branch	**20**	**2**	**0.4**	**18**
CPI			**2.2**	**100**

Composition typique d 'un programme

- **Que se passe-t-il si l'on utilise une mémoire cache plus rapide?**
- **Et si 2 ou 3 instructions ALU pouvaient être exécutées simultanément?**

La loi d ’Amdahl

- Accélération due à l ’amélioration A:

  Accélération(A) = Temps d ’exéc. sans A / Temps d ’exéec. Avec A

- Si A accélère une fraction **F** du programme par un facteur **S**:

  Temps d ’exéc.(avec A) ≤ ((1-F) + F/S) x Temps d ’exéc.(sans A)

  Accélération(avec A) ≤ 1 / ((1-F) + F/S)

- Même si S est grande l ’accélération peut être petite si F est petit…
- Pourquoi ≤ ?

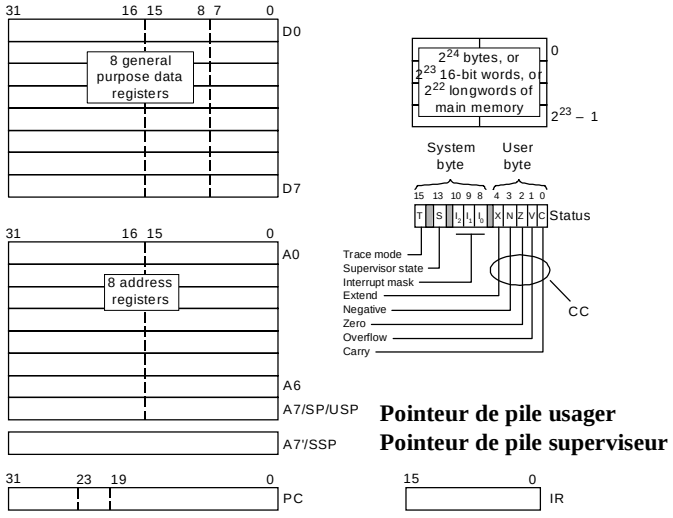
CISC vs. RISC

- CISC: *Complex Instruction Set Computer*
  - Beaucoup d ’instructions et de modes d ’adressage complexes
  - Nombre de cycles / instruction très variable et peut être très grand
  - Pour un compilateur, il n’est pas facile à déterminer le meilleur choix d’instructions pour exécuter un énoncé en langage de haut niveau, difficile à optimiser le code généré
- RISC: *Reduced Instruction Set Computer*
  - Relativement peu d ’instructions simples, peu de modes d’adressage
  - Habituellement un mot par instruction
  - Plusieurs instructions nécessaires pour en émuler une de CISC
  - Calculs d’adresses complexes peut exiger plusieurs instructions
  - Architecture *Load-and-Store*, avec beaucoup de registres généraux

Exemple d'un CISC: Motorola série 68000

- Introduit en 1979
- Un des premier microprocesseurs à 32 bits
  - Majorité d'opérations peut agir sur 32 bits, mais souvent on peut spécifier la taille de l'opérande (8, 16, 32 bits, chaînes de caractères ou de bits)
  - Les chemins de données externes peuvent être de taille différente selon le modèle du processeur (et de l'évolution de la technologie)
    - p.e., MC68000 a eu un bus d'adresses de 24 bits et il y a des modèles avec bus de données de 8 bits ou de 16 bits
- Un CISC typique
  - Grand jeu d 'instructions; opérandes dans des registres et/ou en mémoire
  - 14 modes d'adressage, taille d'instructions varie selon le mode
  - Registres de données et d'adresse, piles superviseur et usager

État du processeur MC68000



État du processeur

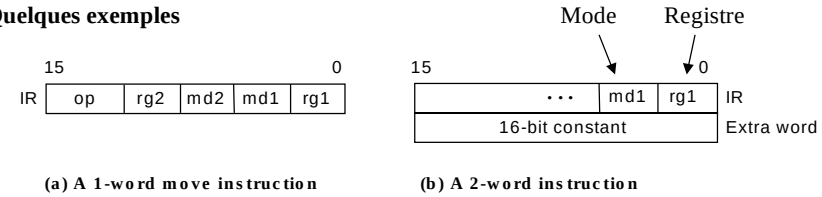
- Distinction entre registres de données et d'adresse
- IR de 16 bits
  - Instructions de taille variable traitées 16 bits à la fois
- Pointeurs de piles
  - D'usager - fait partie des registres d'adresse
  - De système (superviseur) - un registre à part… pourquoi?
- Registre de codes conditionnels (de statut du processeur): un octet de système et de l'usager
  - Arithmétique (N, Z, V, C, X) dans l 'octet d 'usager
  - Statut du système contient les drapeaux des modes superviseur et trace, et le masque d'interruptions

Organisation de la mémoire

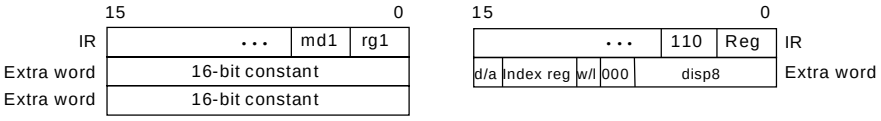
- Le mot (2 octets) et long-mot sont "big-endian"
  - L'octet à l'adresse plus base contient le bit le plus significatif
- Mots et long-mots doivent être alignés sur multiples de 2 resp. de 4 octets
  - Lire la mémoire en la largeur du bus de données et ensuite sélectionner l'octet ou le mot adressé
- Comme autres CISC, une instruction opère sur plusieurs types
  - MOVE.B, MOVE.W, MOVE.L; ADD.B, ADD.W, ADD.L, etc.
  - La longueur d'operande est encodée en bits du mot d'instruction
  - La longueur peut alors être 1, 2 ou 4 octets

Formats d ’instructions

Quelques exemples



(a) A 1-word move instruction **(b) A 2-word instruction**

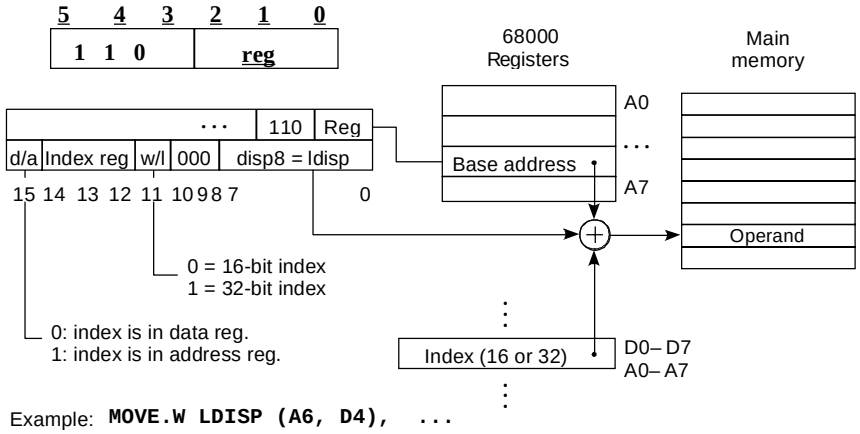


(c) A 3-word instruction **(d) Instruction with indexed address**

Sauf que… champ *registre* peut décrire (étendre) le mode, et certains *modes* décrivent un registre...

Exemple de mode d’adressage complexe

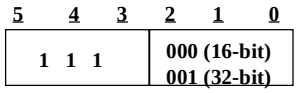
Mode 6: Adresse indexée et avec registre de base

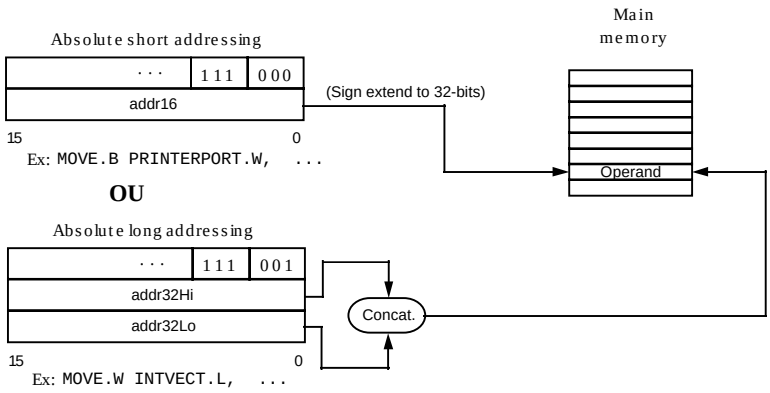


Example: **`MOVE.W LDISP (A6, D4), ...`**

Exemple de mode d'adressage complexe

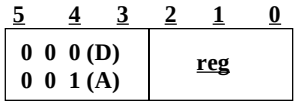
Mode 7: adresse absolue

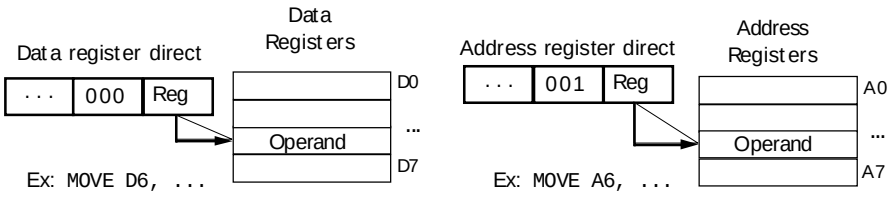




Modes d'adressage simple… aussi

Mode 0 ou 1: Registre direct





- **Et ausi registre indirect, auto-incrément, auto-decrément, registre relatif, relatif indexé, immédiat, etc.**
- **Pas tout mode peut décrire la destination d'un résultat (pas d'adressage relatif)**

Exemples d ’instructions

Inst.	Opérandes	1er mot	XNZVC	Opér.	Taille
MOVE.B	EAs, Ead	0001ddddddssssss	- x x 0 0	dst ← src	byte
MOVE.W	EAs, Ead	0011ddddddssssss	- x x 0 0	dst ← src	word
MOVE.L	EAs, Ead	0010ddddddssssss	- x x 0 0	dst ← src	long
MOVEA.W	EAs, An	0011rrr001ssssss	- - - - -	An← src	word
MOVEA.L	EAs, An	0010rrr001ssssss	- - - - -	An ← src	long
LEA.L	EAc, An	0100aaa111ssssss	- - - - -	An ← EA	addr.
EXG	Dx, Dy	1100xxx1mmmmmyyy	- - - - -	Dx ↔ Dy	long
ADD	EA,Dn	1101rrrmmmaaaaaa	x x x x x	dst ← dst + src	b, w, l
SUB	EA,Dn	1001rrrmmmaaaaaa	x x x x x	dst ← dst - srC	b, w, l
CMP	EA,Dn	1011rrrmmmaaaaaa	- x x x x	dst - src	b, w, l
CMPI	#dat,EA	00001100wwaaaaaa	- x x x x	dst-immed.data	b, w, l
MULS	EA, Dn	1100rrr111aaaaaa	- x x 0 0	Dn←Dn*src	l←w*w

… opérations logiques, test avec 0, décalages, rotations, ...

Exemples d ’instructions - branchements

Op.	Opérandes	mot d ’instr.	Opération
branchement conditionnel:			
Bcc	disp	0110ccccdddddddd DDDDDDDDDDDDDDDD	si (cond) alors PC ← PC + disp
décrémente et branche:			
DBcc	Dn,disp	0101cccc11001rrr	si ¬(cond) alors Dn←Dn-1 si (Dn≠-1) alors PC←PC+disp) sinon PC ← PC + 2
mise à 1 selon condition:			
Scc	EA	0101cccc11aaaaaa	si (cond) alors (EA) ← FFH sinon (EA) ← 00H
sauts inconditionnels:			
JMP	EA	0100111011aaaaaa	PC ← EA
JSR	EA	0100111010aaaaaa	-(SP) ← PC; PC ← EA
RTR		0100111001110111	CC ← (SP)+; PC ← (SP)+
RTS		0100111001110101	PC ← (SP)+
….etc.			

Traitement d'exceptions dans MC68000

(1) Changement du statut du processeur
- Copie temporaire du reg. de statut est fait
- Bit superviseur S est mis à 1, bit de trace T à 0

(2) Vecteur du traitement d'exception
- Décalage du numéro du vecteur (8 bits) à gauche par 2 bits forme une adresse
- Contenu du longue-mot à cette adresse est l'adresse de l'instruction suivante à exécuter - adresse de la routine de service

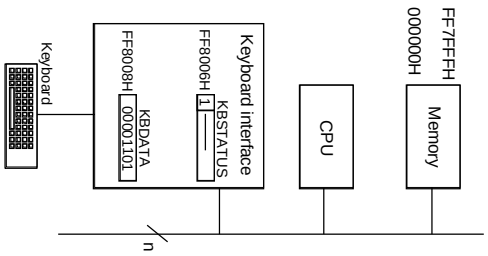
(3) Vieux PC et reg. de statut PS empilés sur la pile superviseur adressée par A7' = SSP

(4) PC est chargé à partir du contenu de l'adresse rangée dans le vecteur

- Retour du traitement par RTE
  - comme RTR mais elle recharge registre de statut PS plutôt que CC
- Priorité du processur (3 bits dans PS) vs. priorité de l'exception

Entrée - Sortie via l'espace d'adresse mémoire

- Un seul bus pour E/S et l'accès à la mémoire
- E/S partage espace d'adresses avec la mémoire et réduit cet espace
  - Demande espace d'adresse plus grand qu'avec un bus E/S séparé
- Le partge de l'espace d'adresses n'est pas fixe
- Habituellement E/S est dans la partie supérieure de l'espace d 'adresses



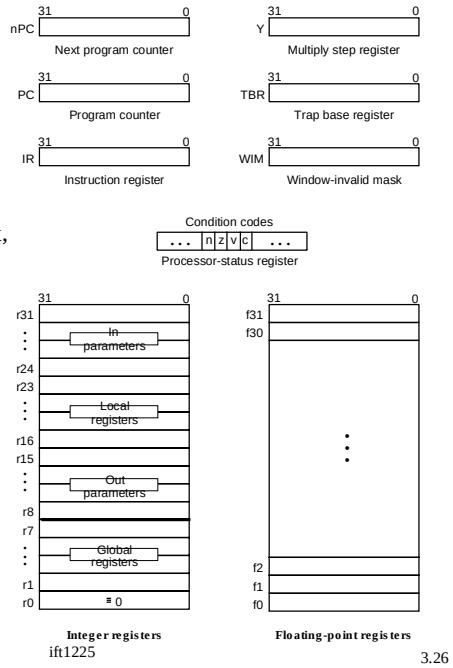
Exemple d'un RISC: SUN SPARC

- Le SPARC possède une architecture "load-and-store" avec un grand nombre des registres généraux, inspiré par le RISC II de UC Berkeley
- Deux modes d'adressage:
  Address = (Reg + Reg) or (Reg + constante de 31 bits)
- Instructions de 32 bits, 69 instructions de base (modèles plus récents +)
- Un jeux de registres à virgule flottante à part des registres généraux
- La 1ère réalisation a eu un pipeline à 4 étages
- Caractéristiques particulières
  - Fenêtre de registres: Sousensembles recouvrants des registres généraux des procédures appelée et appelante pour accélérer le passage de paramètres
  - adresse de 32-bits; organisation de mémoire "big-endian" (vs. "little-endian" du VAX-11)
  - Registre r0 lit toujours 0

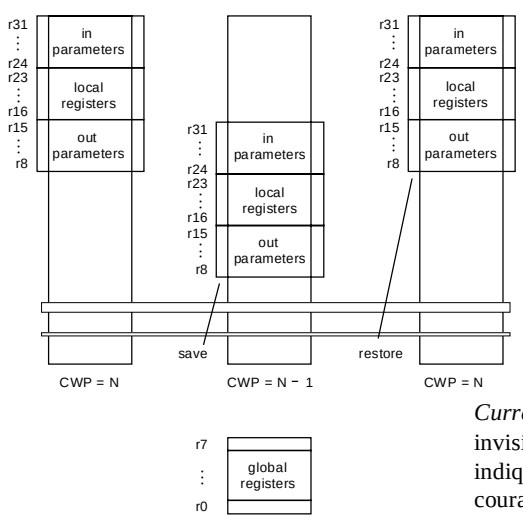
État du processeur (simplifié)

Au total (min) 120 registres généraux, 32 visibles à la fois dans la "fenêtre courante"

Un mot = 32 bits
Un demi-mot = 16 bits
Un octet = 8 bits



Fenêtres des registres



Adr. de retour placée dans r15 par l'appel, se trouve dans r31 après lìnstruction *save*, restauré par *restore*, les deux évoquées dans la procédure appelée

Toute fenêtre utilisée -> un trap dont le service range l'ensemble courant sur la pile en mémoire

Current Window Pointer (CWP) invisible au programmeur indique la position de la fenêtre courante, allocation circulaire

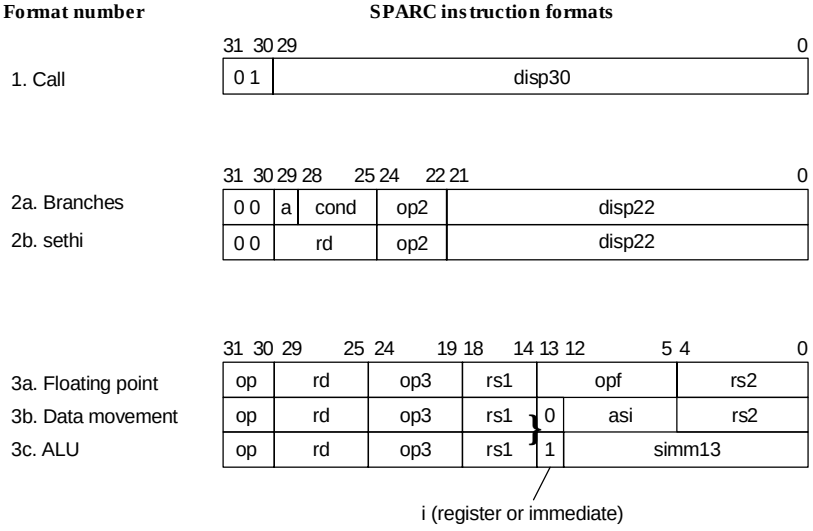
Modes d'adressage

Mode 1: Somme des contenus de deux registres, r0 donne 0

Mode 2: Addition la constante de 13 bits (extension du signe) contenue dans l'instruction au registre spécifié

- Divers forme ainsi obtenues:
  - Indexé: base dans un registre, index dans l'autre
  - Registre indirect: r0 + rn
  - Déplacement: rn + const, n ≠ 0, par +/- 4k octets
  - Absolu : r0 + constant
    (seulement 4k octets en bas ou en haut de la mémoire)

Formats d'instructions



Instructions: Chargement / rangement

Inst.	Op.	OPCODE	explication
ldsb	11	00 1001	Load signed byte
ldsh	11	00 1010	Load signed halfword
ldsw	11	00 1000	Load signed word
ldub	11	00 0001	Load unsigned byte
lduh	11	00 0010	Load unsigned halfword
ldd	11	00 0011	Load doubleword
stb	11	00 0101	Store byte
sth	11	00 0110	Store halfword
stw	11	00 0100	Store word
std	11	00 0111	Store double word
swap	11	00 1111	Swap register with memory
or	10	00 0010	r[d] ← r[s1] OR (r[rs2] or immediate)
sethi	00	Op2=100	High order 22 bits of Rdst ← disp22

Chargement / rangement

- OR utilisé avec r0 pour effectuer un transfert d'un register à un autre
- pour charger un registre par une constante de 32 bits:

  SETHI r17, #supérieurs22

  OR r17, r17, #inférieurs10
- Double-mots chargés dans un registre paire et le suivant impaire
- Les 32 FP registres à virgule flottante peuvent contenir des données de 32, 64 ou 128 bit, représentation standardisée

Arithmétique

Inst.	Op.	OPCODE	explication
add	**10**	**0S 0000**	**Add or add and set condition codes**
addx	**10**	**0S 1000**	**Add with carry: set CCs or not**
sub	**10**	**0S 0100**	**Subtract: subtract and set CCs or not**
subx	**10**	**0S 1100**	**Subtract with borrow: set CCs or not**
mulscc	**10**	**10 1100**	**Do one step of multiply**

- Format 3, Op = 10
- CC positionnées ssi S = 1
- Adressage registre direct ou immédiat

  rd <- rs1 OP rs2 ou rd <- rs1 OP simm13
- Décalage (arithmétique et logique) et op. logiques semblables
- Multiplication, pas à pas, avec MULSCC ou avec des instructions à virgule flottante

Branchements

Inst.	Format	Op	Op2 ou Op3	explication
ba	2	00	010 1000	Unconditional branch
bcc	2	00	010 +4 bits	Conditional branch
call	1	01		Call & save PC in R15
jmpl	3	10	11 1000	Jmp to EA, save PC in Rdst
save	3	10	11 1100	New register window, & ADD
restore	3	10	11 1101	Restore reg. window, & ADD

quelques cc:

Inst.	cond	Inst.	cond	Inst.	cond	Inst.	cond
ba	1000	bne	1001	be	0001	ble	0010
bcc	1101	bcs	0101	bneg	0110	bvc	1111
bvs	0111						

Position de retard ("delay slot") - l'instruction après le branchement…
à être discuté avec l'organisation à pipeline

Exemple d'un programme - appel de procédure

```
        .begin
        .org
prog:   ld      [x], %o0        !Pass parameters in
        ld      [y], %o1        !  first 3 output registers.
        call    add3            !Call subroutine to put result in %o0.
        mov     -17, %o2        !Set last parameter in delay slot
        st      %o0, [z]        !Store returned result.
        ...
x:      15
y:      9
z:      0
add3:   save    %sp,-(16*4),%sp !Get new window and adjust stack pointer.
        add     %i0, %i1, %l0   !Add parameters that now appear in
        add     %l0, %i3, %l0   !  input registers using a local.
        ret                     !Return. Short for jmp %i7+8.
        restore %l0, 0, %o0     !Result moved to caller's %o0.
        .end
```

Exceptions dans SPARC

- Trap = exception interne, Interruption = exception externe
- PSR : 1 bit pour autoriser les traps et 4 bits pour indiquer le niveau d'interruption (PIL) au dessus duquel l'interruption est acceptée
- Exceptions utilisent le système à fenêtres de registres (n'autoriser pas les traps sans avoir la fenêtre suivante libre…):
  - avancer la fenêtre comme avec *save*
  - sauver PC, nPC et PSR dans des regs locaux de la fenêtre
  - inhiber d'autres exceptions
  - transferer le contrôle à la routine de service avec l'aide du registre de base de traps (TBR) et le type de trap de 8 bits
  - ne peut pas modifier les registres in et out de la fenêtre… ils appartiennent au programme interrompu
  - rett restaure la situation…
- SPARC V9 (UltraSPARC est une réalisation) utilise des reg's séparés

Sommaire CISC vs. RISC

- CISC: caractérisé par une économie des instructions et des modes d'adressage - pour occuper moins de mémoire (chère à l'époque)
- Problème si l'on veut accélérer le traitement avec l'aide de pipelines, émission d'instructions multiples, prédiction de branchements, etc.

- RISC: met plus d'accent sur la simplicité et la symmétrie d'instructions, régularité d'exécution, pour faciliter leur exécution et optimisation de programmes par un compilateur.
- Plus compliqué à programmer en langage d'assemblage, car différents astuces sont utlisées pour exploiter le matériel disponible.
- Demande une mémoire plus grande et aussi rapide que le processeur … mémoires caches obligatoires...

Performance ne peut être évaluée qu'avec des programmes qui ressemblent les applications pour lequelles la machine est destinée

MIPS en soi ne dit pas beaucoup… MIPS CISC - MIPS RISC??