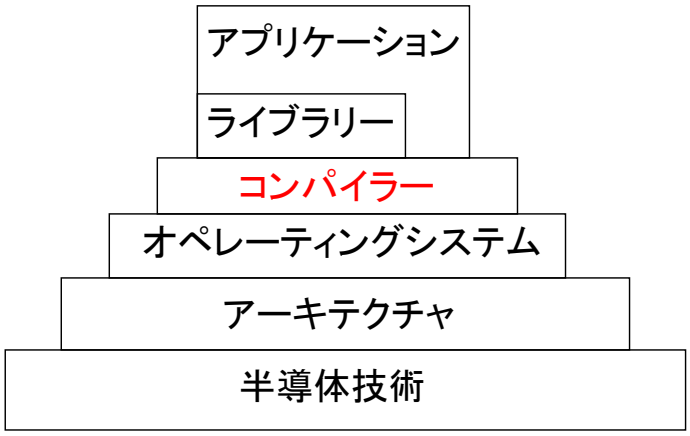
コンパイラの最適化機能により実行時間が短くなる理由

福井　義成

文部科学省　研究振興局
計算科学技術推進室

1

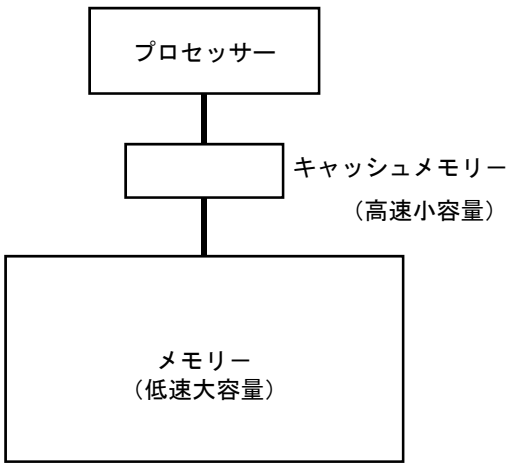
コンパイラーの位置付け



ハードウエアの性質

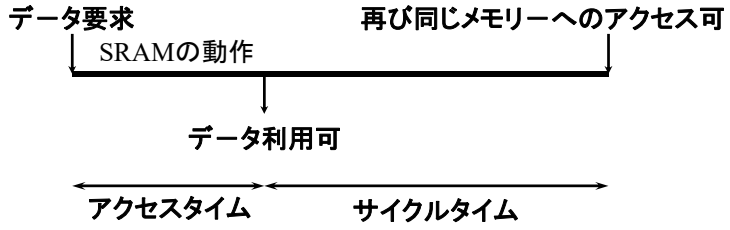
3

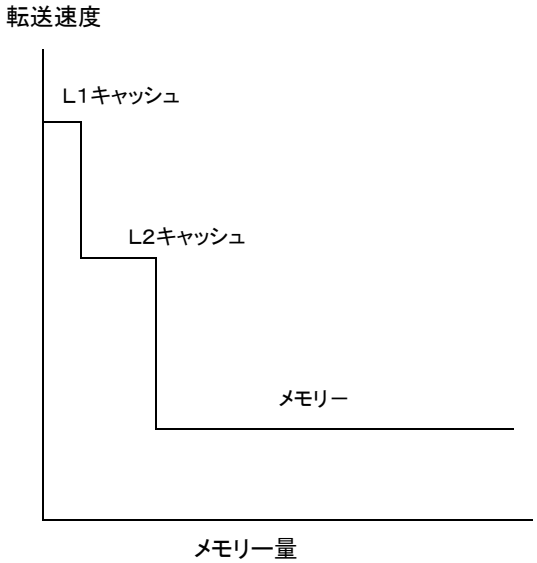
キャッシュメモリーシステム



メインメモリーに使われるDRAMは，SRAMより遅い

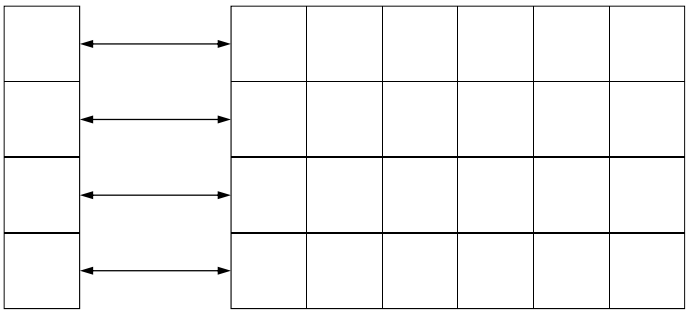
DRAMの動作





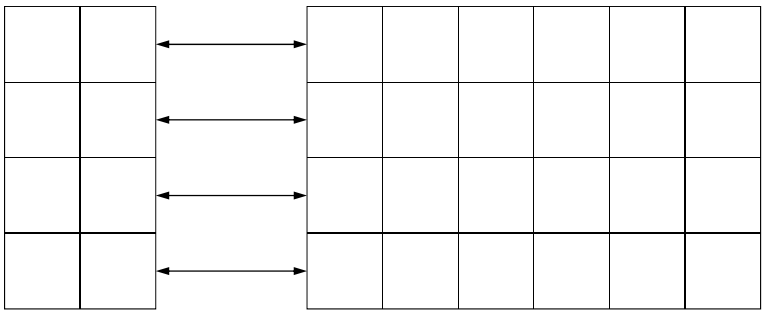
階層型メモリー

直接マップ方式



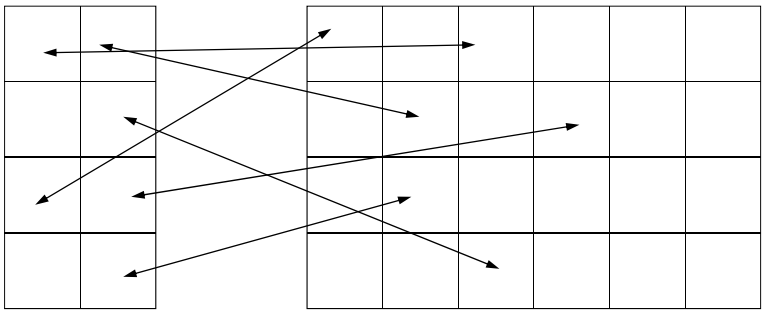
キャッシュ　　メモリー

群連想連想方式(2-way)



キャッシュ　　メモリー

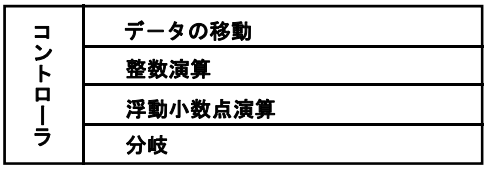
完全連想方式



キャッシュ　　メモリー

スーパースカラー方式

非スーパースカラー方式



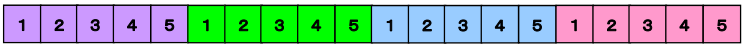
どれか１つの機能のみが実行可能

スーパースカラー方式

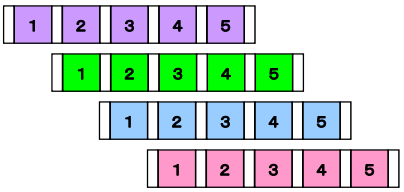
可能ならば４機能が並列に動作

パイプライン化

パイプライン化しない場合



パイプライン化した場合



パイプラインのオーバーヘッド

配列とメモリー上の配置

2次元配列の場合

J →

I ↓

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
(2,1)	(2,2)	(2,3)	(2,4)	(2,5)	(2,6)
(3,1)	(3,2)	(3,3)	(3,4)	(3,5)	(3,6)
(4,1)	(4,2)	(4,3)	(4,4)	(4,5)	(4,6)
(5,1)	(5,2)	(5,3)	(5,4)	(5,5)	(5,6)
(6,1)	(6,2)	(6,3)	(6,4)	(6,5)	(6,6)

FORTRAN

(1,1)	(2,1)	(3,1)	(4,1)	(5,1)	(6,1)

C

(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)

コンパイラーを何処で実行？

- 計算を実行する計算機で行う
- 計算を実行する計算機以外で行う ： クロスコンパイラー
  - 計算を実行する計算機と同種
  - 計算を実行する計算機と異種
    - 同じ機械語の計算機である必要は，無い
      - 良くある計算機
      - 文字処理に向いている計算機
    - （例）ベクトル計算機向けにスカラー計算機で行う
      - ベクトル計算機は，文字処理が苦手

13

コンパイラーの構成

–	プリプロセッサ	前処理	
1.	字句解析	識別子，演算子等の処理	:各言語依存
2.	構文解析	構文を分析	:各言語依存
3.	中間コード生成	言語に依存しない形式化	:言語に共通
4.	コード最適化	コードの最適化	:言語に共通
5.	コード生成	機械語の生成	:言語に共通
–	リンク	ライブラリー等を繋ぐ	:言語に共通
–	実行		

言語のデリミタ等に依存す例（fortranの場合）

do 20 I = 1,20

do 20 I = 1.20　　do20Iという変数に値“1.20” を代入

14

1パスコンパイラーと2パスコンパイラー

- 1パスコンパイラー
  - プログラムを1度でしか読まない
  - 変数宣言は，変数を使う前に必要
  - コンパイル時間が速い
- 2パスコンパイラー
  - プログラムを2度読む
  - 変数宣言は，どこでも良い
  - コンパイル時間が遅い

15

最適化

- opt=0　　最適化をしない
- opt=1　　局所的最適化を行い
- opt=2以上　全体的最適化を行う
- opt=unsafe　計算は，速いが，結果を保証しない

- 最適化の指定方法は，コンパイラーに依存
  - 非常に複雑なものも

16

opt=1の例

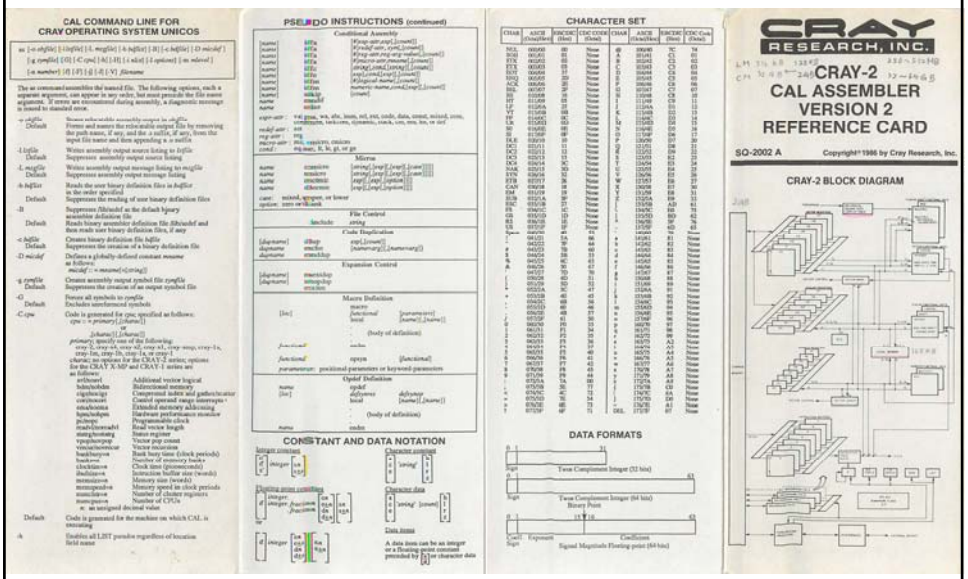
Integer i
i=i*2　　→　　シフト命令

a=b/c/d　　→　　a=b/(c*d)

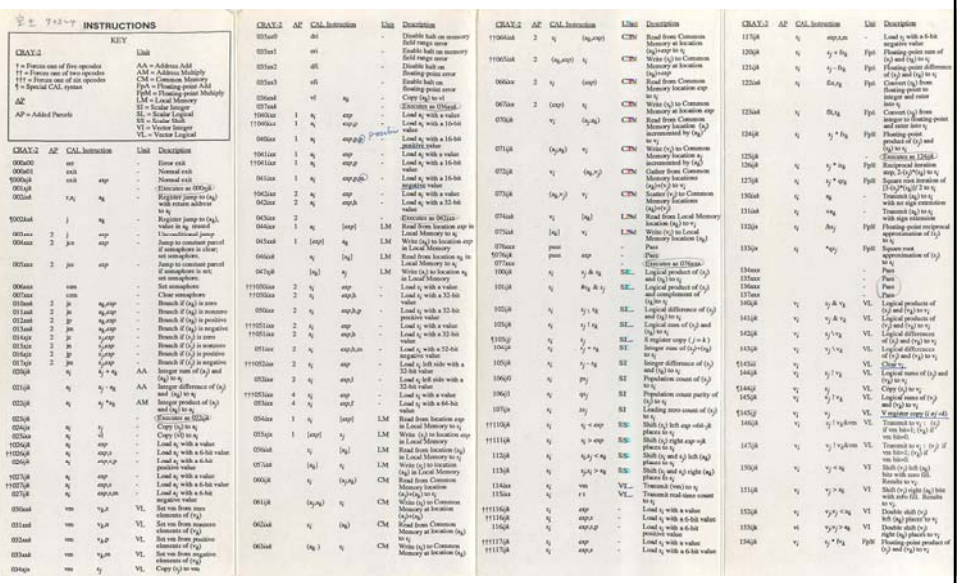
演算を同じ効果があり，計算が速い別の命令に置換える
置換の方法は，計算機によって異なる

17

命令語の例



18



INSTRUCTIONS

KEY

CRAY-1 Unit

† = Forces one of five opcodes — AA = Address Add
†† = Forces one of two opcodes — AM = Address Multiply
††† = Forces one of six opcodes — CM = Common Memory
¶ = Special CAL syntax — FpA = Floating-point Add
FpM = Floating-point Multiply
LM = Local Memory
SI = Scalar Integer
SL = Scalar Logical
SS = Scalar Shift
VI = Vector Integer
VL = Vector Logical

AP
AP = Added Parcels

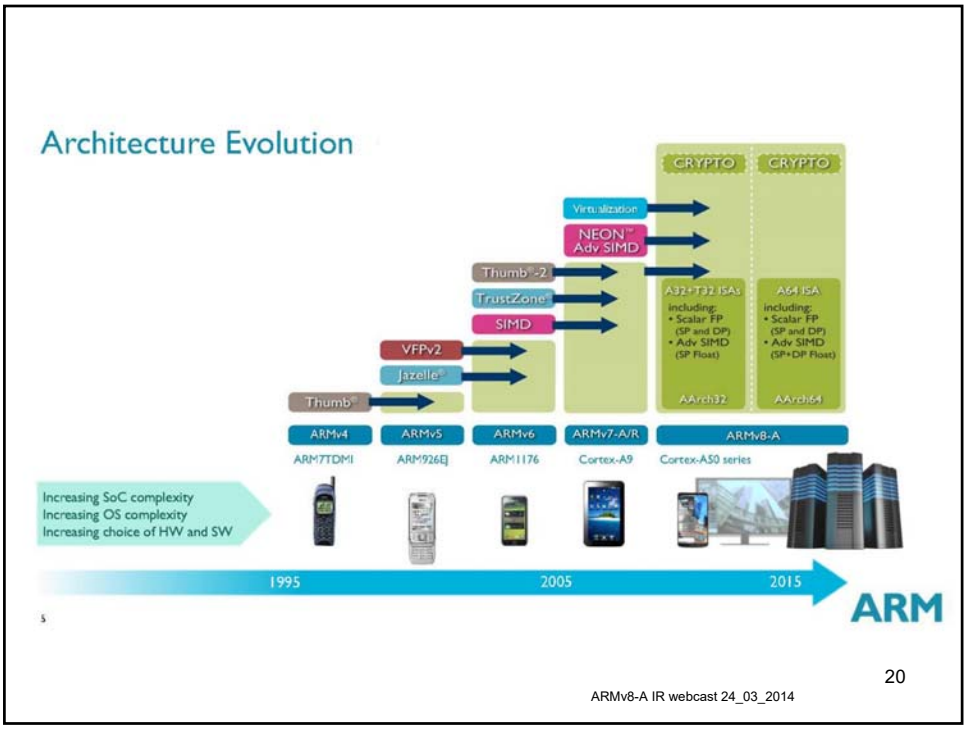
CRAY-2	AP	CAL Instruction	Unit	Description
000x00		err	-	Error exit
000x01		exit	-	Normal exit
¶000xjk		exit	exp	Normal exit
001xjk			-	Executes as 000xjk
002xnk		r_A s_k	-	Register jump to (s_k) with return address to s_j
¶002xnk		j	s_k	-	Register jump to (s_k), value in s_j retained
003xxx	2	j	exp	-	Unconditional jump
004xxx	2	jcn	exp	-	Jump to constant parcel if semaphore is clear; set semaphore.
005xxx	2	jsn	exp	-	Jump to constant parcel if semaphore is set; set semaphore.
006xxx		sem		-	Set semaphore
007xxx		csm		-	Clear semaphore
010xxk	2	jz	s_k,exp	-	Branch if (s_k) is zero
011xxk	2	jn	s_k,exp	-	Branch if (s_k) is nonzero
012xxk	2	jp	s_k,exp	-	Branch if (s_k) is positive
013xxk	2	jm	s_k,exp	-	Branch if (s_k) is negative
014xjx	2	jz	s_j,exp	-	Branch if (s_j) is zero
015xjx	2	jn	s_j,exp	-	Branch if (s_j) is nonzero
016xjx	2	jp	s_j,exp	-	Branch if (s_j) is positive
017xjx	2	jm	s_j,exp	-	Branch if (s_j) is negative
020ijk		s_i	$s_j + s_k$	AA	Integer sum of (s_j) and (s_k) to s_i
021ijk		s_i	$s_j - s_k$	AA	Integer difference of (s_j) and (s_k) to s_i
022ijk		s_i	$s_j * s_k$	AM	Integer product of (s_j) and (s_k) to s_i
023ijk				-	Executes as 022ijk
024ijx		s_i	s_j	-	Copy (s_j) to s_i
025ijx		s_i	-s_j	-	Copy (-s_j) to s_i
1026ijk		s_i	exp	-	Load s_i with a value
1¶026ijk		s_i	exp,i	-	Load s_i with a 6-bit value
026ijk		s_i	exp,+,p	-	Load s_i with a 6-bit positive value
1027ijk		s_i	exp	-	Load s_i with a value
1¶027ijk		s_i	exp,i	-	Load s_i with a 6-bit value
027ijk		s_i	exp,i,m	-	Load s_i with a 6-bit negative value
030ijk		vm	v_j,e	VL	Set vm from zero elements of (v_j)
031ijk		vm	v_j,n	VL	Set vm from nonzero elements of (v_j)
032ijk		vm	v_j,p	VL	Set vm from positive elements of (v_j)
033ijk		vm	v_j,m	VL	Set vm from negative elements of (v_j)
034ijx		vm	v_j	VL	Copy (v_j) to vm

CRAY-2	AP	CAL Instruction	Unit	Description
035xx0		dei	-	Disable halt on memory field range error
035xx1		eei	-	Enable halt on memory field range error
035xx2		dfi	-	Disable halt on floating point error
035xx3		efi	-	Enable halt on floating point error
036xxk		vl	s_k	-	Copy (s_k) to vl
037xxk				-	Executes as 036xxk
1040ijx	1	s_i	exp	-	Load s_i with a value
1¶040ijx	1	s_i	exp,p	-	Load s_i with a 16-bit positive value
1041ijx	1	s_i	exp	-	Load s_i with a value
1¶041ijx	1	s_i	exp,p	-	Load s_i with a 16-bit positive value
041ijx	1	s_i	exp,p,n	-	Load s_i with a 16-bit negative value
1042ijx	2	s_i	exp	-	Load s_i with a value
042ijx	2	s_i	exp,h	-	Load s_i with a 32-bit value
043ijx	2			-	Executes as 042ijx
044ijx	1	s_i	[exp]	LM	Read from location exp in Local Memory to s_i
045xnk	1	[exp]	s_k	LM	Write (s_k) to location exp in Local Memory
046xnk		s_j	[s_k]	LM	Read from location s_k in Local Memory to s_j
047xjk		[s_k]	s_j	LM	Write (s_j) to location s_k in Local Memory
1¶050xjx	2	s_j	exp	-	Load s_j with a value
1¶050xjx	2	s_j	exp,h	-	Load s_j with a 32-bit value
050xjx	2	s_j	exp,h,p	-	Load s_j with a 32-bit positive value
1¶051xjx	2	s_j	exp	-	Load s_j with a value
1¶051xjx	2	s_j	exp,h	-	Load s_j with a 32-bit value
051xjx	2	s_j	exp,h,n	-	Load s_j with a 32-bit negative value
1¶052xjx	2	s_j	exp	-	Load s_j left side with a 32-bit value
052xjx	2	s_j	exp,f	-	Load s_j left side with a 32-bit value
1¶053xjx	4	s_j	exp	-	Load s_j with a value
053xjx	4	s_j	exp,f	-	Load s_j with a 64-bit value
054xjx	1	s_j	[exp]	LM	Read from location exp in Local Memory to s_j
055xjx	1	[exp]	s_j	LM	Write (s_j) to location exp in Local Memory
056xnk		s_j	[s_k]	LM	Read from location (s_k) in Local Memory to s_j
057xnk		[s_k]	s_j	LM	Write (s_j) to location (s_k) in Local Memory
060xjk		s_j	(s_j,s_k)	CM	Read from Common Memory location (s_j)+(s_k) to s_j
061xjk		(s_j,s_k)	s_j	CM	Write (s_j) to Common Memory at location (s_j)+(s_k)
062xnk		s_j	(s_k)	CM	Read from Common Memory at location (s_k) to s_j
063xnk		(s_k)	s_j	CM	Write (s_j) to Common Memory at location (s_k)

CRAY-2	AP	CAL Instruction	Unit	Description
1¶064xnk	2	s_j	(s_k,exp)	CIN	Read from Common Memory at location (s_k)+exp to s_j
1¶065xnk	2	(s_k,exp)	s_j	CIN	Write (s_j) to Common Memory at location (s_k)+exp
066xnx	2	s_j	(exp)	CIN	Read from Common Memory location exp to s_j
067xnx	2	(exp)	s_j	CIN	Write (s_j) to Common Memory at location exp
070xjk		s_j	(s_j,s_k)	CIN	Read from Common Memory location (s_j) incremented by (s_k)[1] to s_j
071xjk		(s_j,s_k)	s_j	CIN	Write (s_j) to Common Memory location s_j incremented by (s_k)[1]
072xjk		s_j	(s_k,p)	CIN	Gather from Common Memory locations (s_k)+(s_j) to s_j
073xjk		(s_k,p)	s_j	CIN	Scatter (s_j) to Common Memory locations (s_k)+(s_j)
074xnk		s_j	[s_k]	LIN	Read from Local Memory location (s_k) to s_j
075xnk		[s_k]	s_j	LIN	Write (s_j) to Local Memory location (s_k)
076xxx		pass		-	Pass
¶076xjk		pass	exp	-	Pass
077xxx				-	Executes as 076xxx
100xjk		s_j	s_j & s_k	SL	Logical product of (s_j) and (s_k) to s_j
101xjk		s_j	s_k & s_j	SL	Logical product of (s_j) and complement of (s_k) to s_j
102xjk		s_j	s_j | s_k	SL	Logical difference of (s_j) and (s_k) to s_j
103xjk		s_j	s_j ! s_k	SL	Logical sum of (s_j) and (s_k) to s_j
¶103xjj		s_j	s_j	SL	0 register copy ($j = k$)
104xjk		s_j	s_j + s_k	SI	Integer sum of (s_j)+(s_k) to s_j
105xjk		s_j	s_j - s_k	SI	Integer difference of (s_j) and (s_k) to s_j
106x0		s_j	pop	SI	Population count of (s_j) to s_j
106x1		s_j	qpy	SI	Population count parity of (s_j) to s_j
107xjx		s_j	zcy	SI	Leading zero count of (s_j) to s_j
1¶110xjk		s_j	s_j < exp	SS	Shift (s_j) left exp +64 ,& places to s_j
1¶111xjk		s_j	s_j > exp	SS	Shift (s_j) right exp -n,& places to s_j
112xjk		s_j	$s_j s_j$ < s_k	SS	Shift (s_j and s_j) left (s_k) places to s_j
113xjk		s_j	$s_j s_j$ > s_k	SS	Shift (s_j and s_j) right (s_k) places to s_j
114xxx		s_j	vm	VL	Transmit (vm) to s_j
115xxx		s_j	r t	VL	Transmit real-time count to s_j
1¶116xjk		s_j	exp	-	Load s_j with a value
1¶116xjk		s_j	exp,i	-	Load s_j with a 6-bit value
116xjk		s_j	exp,i,p	-	Load s_j with a 6-bit positive value
1¶117xjk		s_j	exp	-	Load s_j with a value
1¶117xjk		s_j	exp,i	-	Load s_j with a 6-bit value

CRAY-2	AP	CAL Instruction	Unit	Description
117xjk		s_j	exp,i,m	-	Load s_j with a 6-bit negative value
120xjk		s_j	s_j + s_k	FpA	Floating-point sum of (s_j) and (s_k) to s_j
121xjk		s_j	s_j - s_k	FpA	Floating-point difference of (s_j) and (s_k) to s_j
122xnk		s_j	float$_k$	FpA	Convert (s_k) from floating point to integer and enter into s_j
123xnk		s_j	fix$_k$	FpA	Convert (s_k) from integer to floating point and enter into s_j
124xjk		s_j	s_j * s_k	FpM	Floating-point product of (s_j) and (s_k) to s_j
125xjk				-	Executes as 126xjk
126xjk		s_j	s_j * s_k	FpM	Reciprocal iteration step, 2-(s_j*s_k) to s_j
127xjk		s_j	s_j * qrs	FpM	Square root iteration of (3-(s_j)*(s_k))/ 2 to s_j
130xnk		s_j	s_k	-	Transmit (s_k) to s_j with no sign extension
131xnk		s_j	+s_k	-	Transmit (s_k) to s_j with sign extension
132xjx		s_j	/s_j	FpM	Floating-point reciprocal approximation of (s_j) to s_j
133xjx		s_j	*qrj	FpM	Square root approximation of (s_j) to s_j
134xxx				-	Pass
135xxx				-	Pass
136xxx				-	Pass
137xxx				-	Pass
140xjk		v_i	s_j & v_k	VL	Logical products of (s_j) and (v_k) to v_i
141xjk		v_i	s_j & v_k	VL	Logical products of (s_j) and (v_k) to v_i
142xjk		v_i	s_j \ v_k	VL	Logical differences of (s_j) and (v_k) to v_i
143xjk		v_i	s_j \ v_k	VL	Logical differences of (s_j) and (v_k) to v_i
¶145xi		v_i		VL	Clear v_i.
146xjk		v_i	s_j ! v_k	VL	Logical sums of (s_j) and (v_k) to v_i
¶146xjj		v_i	s_j	VL	Copy (s_j) to v_i
145xjk		v_i	s_j ! v_k	VL	Logical sums of (s_j) and (v_k) to v_i
¶145xjj		v_i	v_j	VL	Leading zero count of (s_j) to v_i / V register copy (i ≠j =k)
146xjk		v_i	s_j ! v_k&vm	VL	Transmit to v_i (s_j) if vm bits=1; (v_k) if vm bits=0.
147xjk		v_i	v_j ! v_k&vm	VL	Transmit to v_i (v_j) if vm bits=1; (v_k) if vm bits=0.
150xjk		v_i	v_j < s_k	VI	Shift (v_j) left (s_k) bits with zero fill. Results to v_i
151xjk		v_i	v_j > s_k	VI	Shift (v_j) right (s_k) bits with zero fill. Results to v_i
152xjk		v_i	$v_j s_j$ < s_k	VI	Double shift (v_j) left (s_k) places to v_i
153xjk		v_i	$v_j s_j$ > s_k	VI	Double shift (v_j) right (s_k) placed to v_i
154xjk		v_i	s_j * fx_k	FpM	Floating-point product of (s_j) and (v_k) to v_i

19



20

opt=2の例

```
do  i=1,n                tmp=c/d
    a(i)=b(i)*(c/d)      do i=1,n
enddo                        a(i)=b(i)*tmp
                         enddo
```

基本的に，変化の無い項をループの外に移動する
ループの反復回数が少ないと遅くなる

論理演算の場合，より演算数が少ない別の命令列に置換できる場合がある
　LSIの故障シミュレータの例
　計算機の命令の速度に依存

21

計算順序が変わった場合

```
 1234567890123.4
+)           1.2345678901234

 1234567890124.6

-1234567890123.3
+)           2.3436789012345

-1234567890121.0

      add→ 3.6000000000000
           ^^^^^^^^^^^^
```

```
 1234567890123.4
-1234567890123.3
             1.2345678901234
+)           2.3436789012345

             3.6782467913579
```

並列化，ベクトル化の場合，
注意が必要

桁落ちの原因

- 絶対値が近い2つの数値の「減算」で桁落ちを起こす
  - 入力に依存する
- 桁落ちの原因
  - 物理定数の差による
    - 構造解析　：10^{-5}
    - CCDの解析　：10^{-13}
      - 計算機により，解が得られない場合と得られる場合がある
  - メッシュの細分化による連立一次方程式の依存関係の増加

計算結果が異なる例

a=b/c/d　→　a=b/(c*d)

EISPACKで，上記のように変更すると計算できくなる.
(c*d)の値が，オーバーフローを起こす

24

コンパイラーは，どこまでやるべきか？

- 高度の最適化は，演算時間を短縮化できる可能性がある
- しかし，コンパイル時間がかかる
  - 日立のIAPの初期の例
    - s=s+a(i)*b(i)　：ベクトル化
    - s=a(i)*b(i)+s　：ベクトル化できない
      - 構文解析が未熟
      - でもこれは，必要でしょうか？

25

高速化の簡単な例

$$S = \sum_{I=1}^{10000} (-1)^i \cdot i$$

高速化のレベル

1. ムダな計算を削除する
2. コーディングを変更する
3. 計算を行う場合のデータ構造を変更する
4. 問題を解くアルゴリズムを変更
5. 問題のモデル化そのものを変更

プログラム　1

```
S＝0. 0
DO  I＝1, 10000
  S＝S＋(－1)＊＊I  ＊  I
CONTINUE
```

$$S = \sum_{I=1}^{10000} (-1)^i \cdot i$$

プログラム　2

```
S＝0. 0
A＝1. 0
B＝0. 0
DO  I＝1, 10000
    A＝－A
    S＝S＋A＊I
CONTINUE
```

プログラム　3

```
S＝0. 0
A＝1. 0
B＝0. 0
DO  I＝1, 10000
    A＝－A
    B＝B＋1. 0
    S＝S＋A＊B
CONTINUE
```

プログラム　4

```
S＝0. 0
A＝－1. 0
DO  I＝1, 10000, 2
   A＝A＋2. 0
   S＝S－A
CONTINUE
A＝0. 0
DO  I＝2, 10000, 2
   A＝A＋2. 0
   S＝S＋A
CONTINUE
```

プログラム　５

S＝N／2
IF(MOD(N, 2). EQ. 1)S＝S－N

```
(P－1)
      S=0. 0
      DO  20  I=1, 10000
        S=S+(－1)＊＊I ＊ I
   20  CONTINUE

(P－2)
      S=0. 0
      A=1. 0
      DO  I=1, 10000
        A=－A
        S=S+A＊I
      CONTINUE

      (P－5)
            N=10000
            S=N／2
            IF(MOD(N, 2) . EQ.  1)  S=S－N
```

```
(P－3)
      S=0. 0
      A=1. 0
      B=0. 0
      DO  I=1, 10000
        A=－A
        B=B+1. 0
        S=S+A＊B
      CONTINUE

(P－4)
      S=0. 0
      A=－1. 0
      DO  I=1, 10000, 2
        A=A+2. 0
        S=S－A
      CONTINUE
      A=0. 0
      DO  I=2, 10000, 2
        A=A+2. 0
        S=S+A
      CONTINUE
```

計算時間

① 31. 517811 m秒
② 10. 800622 m秒
③ 7. 770627 m秒
④ 6. 225157 m秒
⑤ 0. 003281 m秒

ご清聴ありがとうございます

34