情報科学実験C　中間レポート

氏名　山久保孝亮
所属　大阪大学基礎工学部情報科学科ソフトウェア科学コース
メールアドレス　u327468b@ecs.osaka-u.ac.jp
学籍番号　09B22084

提出日　2024 年 12 月 12 日

1　Tiny-Processor と C-Processor のデータパスの違い

以下の図 1,2 はそれぞれ Tiny-Processor と C-Processor のデータパスである．

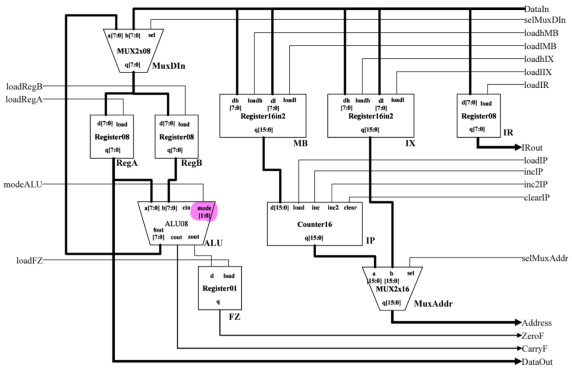

図 1: Tiny-Processor のデータパス

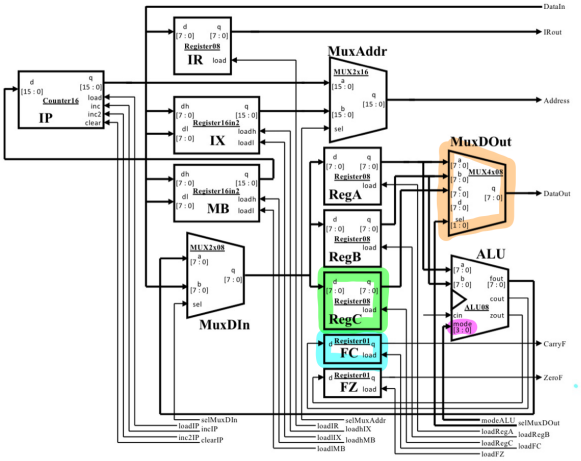

図 2: C-Processor のデータパス

この二つのデータパスの違いを明確にするために色付けをした．以下でそれぞれの違いについて述べる．

- 図 1 のピンクに塗った部分は ALU の入力 mode を表す．Tiny-Processor では mode は 2 ビットの入力として設計されている．図 2 のピンクで塗った部分は C-Processor の同じく ALU の入力 mode である．C-Processor では Tiny-Processor よりも機能が増えており，mode の入力が 4 ビットに増加している．

- 図 2 の緑で塗った部分は RegC であり，C-Processor で追加されている．入力は新たに追加された loadRegC 以外は RegA,RegB と同じである．このレジスタ C は STDI 命令の際に使用される．詳細な説明は 2 で記述する．

- 図 2 の青色で塗った部分は FC であり，C-Processor で追加されている．入力は新たに追加された loadFC と ALU の出力である cout で，出力は q 即ち CarryF である．これは ALU の演算の結果，桁上りが発生したかを判定する．これから CarryF という信号が得られ，JPC 命令に利用される．

- 図 2 のオレンジで塗った部分は MuxDOut であり，4 入力のマルチプレクサである．ただし，4 入力の内 3 つしか使っておらずその 3 つの入力は RegA,RegB,RegC の出力である．また，4 つの入力の内どの値を出力するかを決定する 2 ビットの selMuxDOut を追加した．そして出力が DataOut となる．Tiny-Processor では RegA の出力が DataOut となっていたが，C-Processor では三つのレジスタからマルチプレクサで選択して DataOut を決定している．

2　STDI 命令のデータの流れ

以下の図 3 は 1 の図 2 で示した C-Processor のデータパスに，STDI 命令を実行する際のデータの流れを追加したものである．

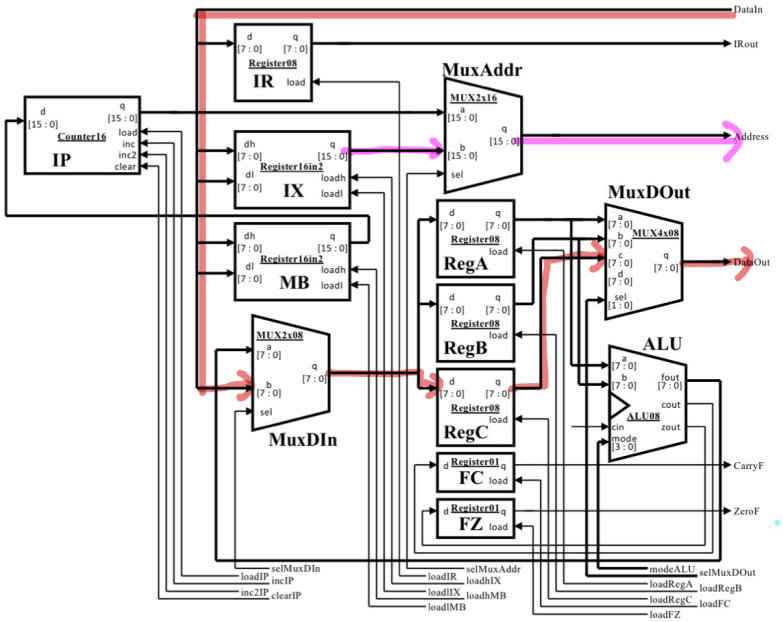

図 3: C-Processor における STDI 命令のデータの流れ

ピンクの矢印はアドレス，赤の矢印はデータの流れを表している．STDI 命令は以下の流れで実行される．

1. STDI 命令が実行される前に IX にアドレスが格納される．
2. DataIn からデータが読み込まれ，レジスタ C にその値が格納される．
3. MuxDOut によって c が選択され，また MuxAddr によって b が選択されることによって IX に入っていたアドレスに DataIn の値が即値で格納される．

3　外部出力信号用ジョンソンカウンタと内部制御信号用ジョンソンカウンタ

3.1　外部出力用ジョンソンカウンタ

C-Processor の外部出力用ジョンソンカウンタは以下のようになる．

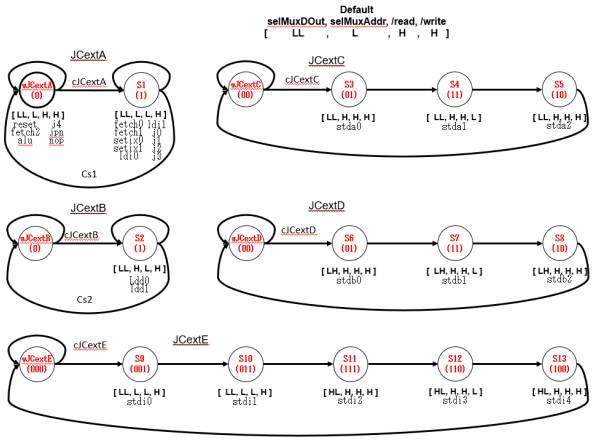

図 4: 外部出力用ジョンソンカウンタ

外部出力用ジョンソンカウンタはメモリのアクセスに関係する信号を制御する．Tiny-Processor から，STDB 命令と STDI 命令を制御するためのジョンソンカウンタを追加した．そして selMuxDOut，selMuxAddr，/read，/write の 4 つの信号の値をそれぞれの状態に合わせて変動した．

3.2　内部出力用ジョンソンカウンタ

C-Processor の内部出力用ジョンソンカウンタは以下のようになる．

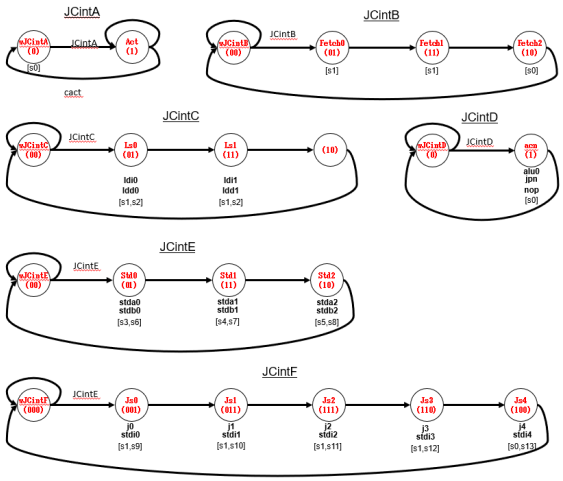

図 5: 内部出力用ジョンソンカウンタ

内部出力用ジョンソンカウンタはレジスタのロード信号やマルチプレクサの選択信号を制御する．命令の

種類即ち IR の値と内部出力用ジョンソンカウンタの値を用いて，外部出力脱出用信号，内部出力脱出用信号，脱出信号以外の条件を作成した．以下はその条件を表す．

信号名	デフォルト値	出力値	内部状態	条件
cJCextA	L	H	reset	qJCintA = '0'
			fetch2	qJCintB = "10" and IR = (SETIXH,SETIXL,LDIA,LDIB,JP,JPZ(Z=1),JPC(C=1))
			acn	qJCintD = '1'
			js4	qJCintF = "100"
			ls1	qJCintC = "11" and IR = (LDDA,LDDB)
			std2	qJCintE = "10"
cJCextB	L	H	fetch2	qJCintB = "10" and IR = (LDDA,LDDB)
cJCextC	L	H	fetch2	qJCintB = "10" and IR = (STDA)
cJCextD	L	H	fetch2	qJCintB = '10' and IR = (STDB)
cJCextE	L	H	fetch2	qJCintB = "10" and IR = (STDI)
cs1	L	H	fetch1	qJCintB = "11"
			js3	qJCintF = "110"
cs2	L	H	ls1	qJCintC = "11" and IR = (LDDA,LDDB)

図 6: 外部出力脱出信号の条件

信号名	デフォルト値	出力値	内部状態	条件
cact	L			ソフトウェアリセットを考えないので常にL
cJCintB	L	H	reset	qJCintA='0'
			ls1	qJCintC=11
			can	qJCintD=1
			std2	qJCintE=10
			js4	qJCintF=100
cJCintC	L	H	fetch2	qJCintB = 10 and IR = (SETIXH,SETIXL,LDIA,LDIB,LDDA,LDDB)
cJCintD	L	H	fetch2	qJCintB=10 and IR = (ADDA,ADDB,SUBA,SUBB,ANDA,ANDB,ORA,ORB,NOTA,NOTB,INCA,INCB,DECA,DECB,CMP,NOP,JPZ(Z=0),JPC(C=0))
cJCintE	L	H	fetch2	qJCintB=10 and IR = (STDA,STDB)
cJCintF	L	H	fetch2	qJCintB=10 and IR = (JP,JPZ(Z=1),JPC(C=1),STDI)

図 7: 内部出力脱出信号の条件

信号名	デフォルト値	出力値	内部状態	条件
clearIP	L	H	reset	qJCintA=0
loadIR	L	H	fetch1	qJCintB=11
modeALU	LLLL	**	acn	qJCintD=1 and IR = (ADDA,ADDB,SUBA,SUBB,ANDA,ANDB,ORA,ORB,NOTA,NOTB,INCA,INCB,DECA,DECB,CMP)
loadFZ	L	H	acn	qJCintD=1 and IR = (ADDA,ADDB,SUBA,SUBB,ANDA,ANDB,ORA,ORB,NOTA,NOTB,INCA,INCB,DECA,DECB,CMP)
loadFC	L	H	acn	qJCintD=1 and IR = (ADDA,ADDB,SUBA,SUBB,ANDA,ANDB,ORA,ORB,NOTA,NOTB,INCA,INCB,DECA,DECB,CMP)
loadhMB	L	H	js1	qJCintF=011 and IR = (JP,JPZ(Z=1),JPC(C=1))
loadlMB	L	H	js3	qJCintF=110 and IR = (JP,JPZ(Z=1),JPC(C=1))
loadIP	L	H	js4	qJCintF=100 and IR = (JP,JPZ(Z=1),JPC(C=1))
loadhIX	L	H	ls1	qJCintC=11 and IR = (SETIXH)
loadlIX	L	H	ls1	qJCintC=11 and IR = (SETIXL)
loadRegA	L	H	ls1	qJCintC=11 and IR = (LDIA,LDDA,)
			acn	qJCintD=1 and IR = (ADDA,SUBA,ANDA,ORA,NOTA,INCA,DECA)
loadRegB	L	H	ls1	qJCintC=11 and IR = (LDIB,LDDB)
			acn	qJCintD=1 and IR = (ADDB,SUBB,ANDB,ORB,NOTB,INCB,DECB)
loadRegC	L	H	js1	qJCintF=011 and IR = (STDI)
incIP	L	H	fetch2	qJCintB=10
			ls1	qJCintC=11 and IR = (SETIXH,SETIXL,LDIA,LDIB)
			js1	qJCintF=011
inc2IP	L	H	acn	qJCintD=1 and IR = (JPZ(Z=0),JPC(C=0))
selMuxDIn	L	H	ls0	qJCintC=01 and IR = (LDIA,LDIB,LDDA,LDDB)
			ls1	qJCintC=11 and IR = (LDIA,LDIB,LDDA,LDDB)
			js0	qJCintF=001 and IR = (STDI)
			js1	qJCintF=011 and IR = (STDI)

図 8: 脱出信号以外の条件

4　拡張課題

4.1　拡張課題 a：プロセッサ命令セットの拡張

今回私が追加した命令セットは論理シフトである．いかにその仕様を示す．この 4 命令は，第一オペラン

命令	オペランド	サイズ	コード
SLLA	id	1 word	11000000
SRLA	id	1 word	11000001
SLLB	id	1 word	11001000
SRLB	id	1 word	11001001

表 1: 追加した命令

ドで即値により指定した数値分レジスタ A またはレジスタ B を論理シフトするものである．これは ALU の演算の処理を切り替える 4 ビットのベクトル modeALU で未使用の部分を使用して実装した．したがって，このシフト命令を追加した後の modeALU の処理は以下のようになる．

LLLL	A + B	LHLL	not A	HLLL	not B	HHLL	sllb
LLLH	A - B	LHLH	A + 1	HLLH	B + 1	HHLH	srlb
LLHL	A and B	LHHL	A - 1	HLHL	B - 1	HHHL	未使用
LLHH	A or B	LHHH	slla	HLHH	srla	HHHH	未使用

表 2: シフト命令を追加した後の modeALU の処理

また，ALU が即値を扱えるようにするために入力としてレジスタ C を追加した．追加後のデータパスアーキテクチャ図は以下のようになる．赤色の矢印が新たに追加したレジスタ C から ALU への入力である．

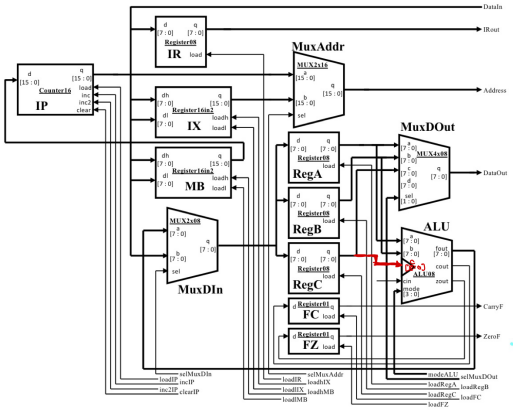

図 9: 追加後のデータパスアーキテクチャ図

また，SLLA,SRLA の外部出力用ジョンソンカウンタとして wJextF を，SLLB,SRLB 用の外部出力用ジョンソンカウンタとして wJextG を追加した．これらはどちらも 2 ビットのジョンソンカウンタで実装した．したがって，追加後の外部出力用のジョンソンカウンタは以下のようになる．

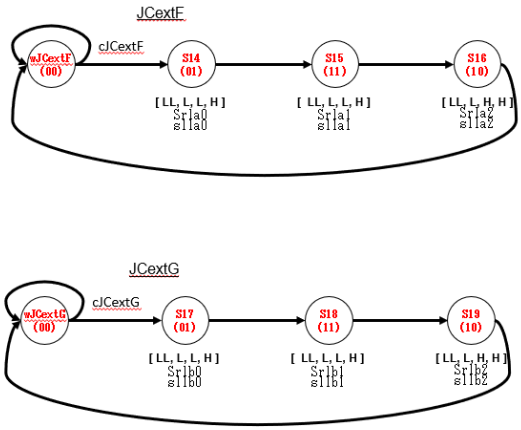

図 10: 追加後の内部出力用ジョンソンカウンタ

そしてメモリアクセス用の信号の条件に上記の内容を追加した．また，内部出力用のジョンソンカウンタは新たに追加せず JCintC を利用して実装した．即ち，追加したシフト命令は cJCintC は LDIA,LDIB と同じ内部出力用ジョンソンカウンタを使用して脱出信号等を制御した．したがって，図 6 から図 8 の条件にさらにシフト命令用の条件を追加した．
具体的なシフトの処理は，NUMETRIC_STD ライブラリを使用し，8 ビットのベクトルである c を整数に変換し，それを使用して a または b を論理シフトするという方法で実装した．以下では実行結果を示す．

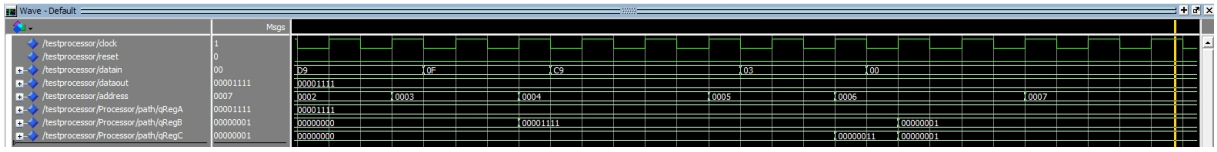

図 11: シフト命令の実行結果

これは，レジスタ A,B のどちらにも 00001111 が格納されている際に”SRLB 03”を実行した際の波形である．address が 0006 の際にレジスタ C に 3 を表す 00000011 が入り，その後 B が右に 3 ビットシフトされてレジスタ B に 00000001 が格納されている．

4.2 拡張課題 b：演算機の改善

1. まず，桁上げ先見加算器について説明する．桁上げ先見加算器は最大の遅延が O(logn) の加算器であり，演算対象の桁数が多い場合に効率よく計算が実行できる．今回はまず 4 ビットの桁上げ先見加算器について記述し，それを組み合わせて 8 ビット，16 ビットの加算器を作成する．
   桁上げ先見加算器はまず入力 x と y に対して桁上げ生成関数 G と桁上げ伝播関数 P を計算する．計算式は以下のようになる．

$$g_i = x_i \wedge y_i$$

$$p_i = x_i \oplus y_i$$

   i は 4 ビットの内の何桁目かを表しており，例えば g_2 は下から三桁目を表している．そして，これらの値を用いて以下のように出力 s と桁上げ出力 c を計算する．

$$s_i = p_i \oplus c_{i-1}$$

$$c_i = g_i \vee (p_i \wedge c_{i-1})$$

この漸化式に代入を繰り返していくと，最終的に x_i と y_i と c_{-1} のみで表すことができるようになる．したがって，RCA とは違い，下位ビットの桁上りを待つ必要がなくなる．

2. 今回は上記の方法でまず 4 ビットの LCA を設計した．そして，それを二つ利用して 8 ビットの LCA を実現した．具体的には，一つの LCA には入力の下位 4 ビット，c_{-1} を入力とし，もう一つの LCA には入力の上位 4 ビット，下位ビット用の LCA が出力したキャリーを入力とした．そして最後に結果を結合することにより計算を実現した．16bit 桁上げ先見加算器も 8 ビット LCA に対して同様にすることで実装した．

3.

5 感想

今回の課題を通して CPU がどのように設計されているのかについて学ぶことができて非常に興味深かった．特に，内部出力用ジョンソンカウンタと外部出力用ジョンソンカウンタを分けて信号を制御している部分がとても驚いた．これまでの実験を通して問題を分割することの重要性を再三認識してきたが，また改めてその重要性を感じることができた．