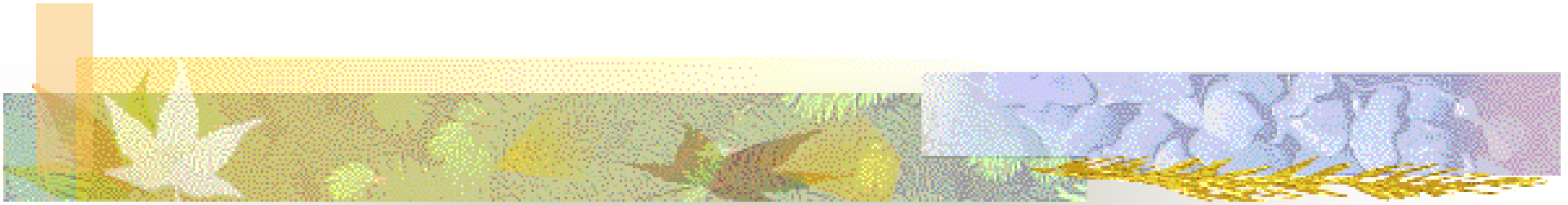
北海道大学
HOKKAIDO UNIVERSITY

コンピュータシステム
（アーキテクチャ　第２回）



工学部 情報エレクトロニクス学科
大学院 情報科学研究院 情報理工学部門
堀山 貴史

講義資料について

- アーキテクチャの回では教科書は指定しません。
- 講義資料は下記のURLからダウンロードしてください。
  - 毎週の講義スライドを置いておきます。

    https://art.ist.hokudai.ac.jp/~horiyama/comp_sys/

- 参考書
  - 雨宮真人, 田中譲:「コンピュータアーキテクチャ」, オーム社
    （旧コースの教科書。昭和時代の本。絶版で入手困難）
  - パターソン＆ヘネシー:「コンピュータの構成と設計 第5版（上・下）」,
    日経BP社（名著だが、上下巻合わせて約9000円する。高価）

前回(アーキテクチャ第1回)の内容

計算機アーキテクチャとは

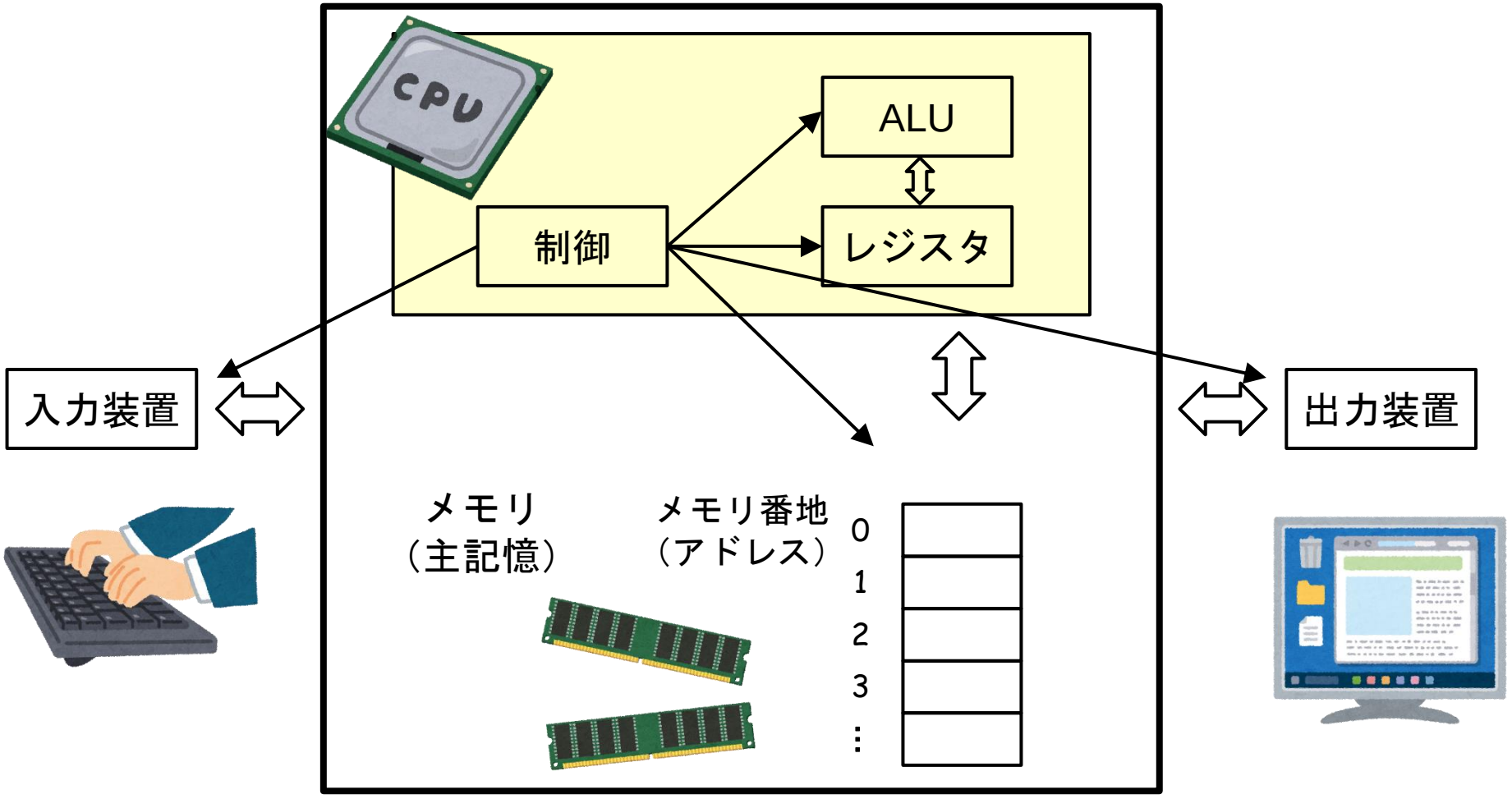
- 現代社会におけるコンピュータシステム
- 計算機の利用形態の分類とその特性
- アーキテクチャの評価尺度とトレードオフ
- ハードウェアの構成要素
  - レジスタ、演算器、プログラムカウンタ、命令デコーダ、メモリ(主記憶)、バス、ディスク(2次記憶)、入出力装置
- ソフトウェアの構成要素
  - 機械語、ファームウェア、オペレーティングシステム、コンパイラ/インタプリタ、ソフトウェアライブラリ、アプリケーションプログラム

今回の内容

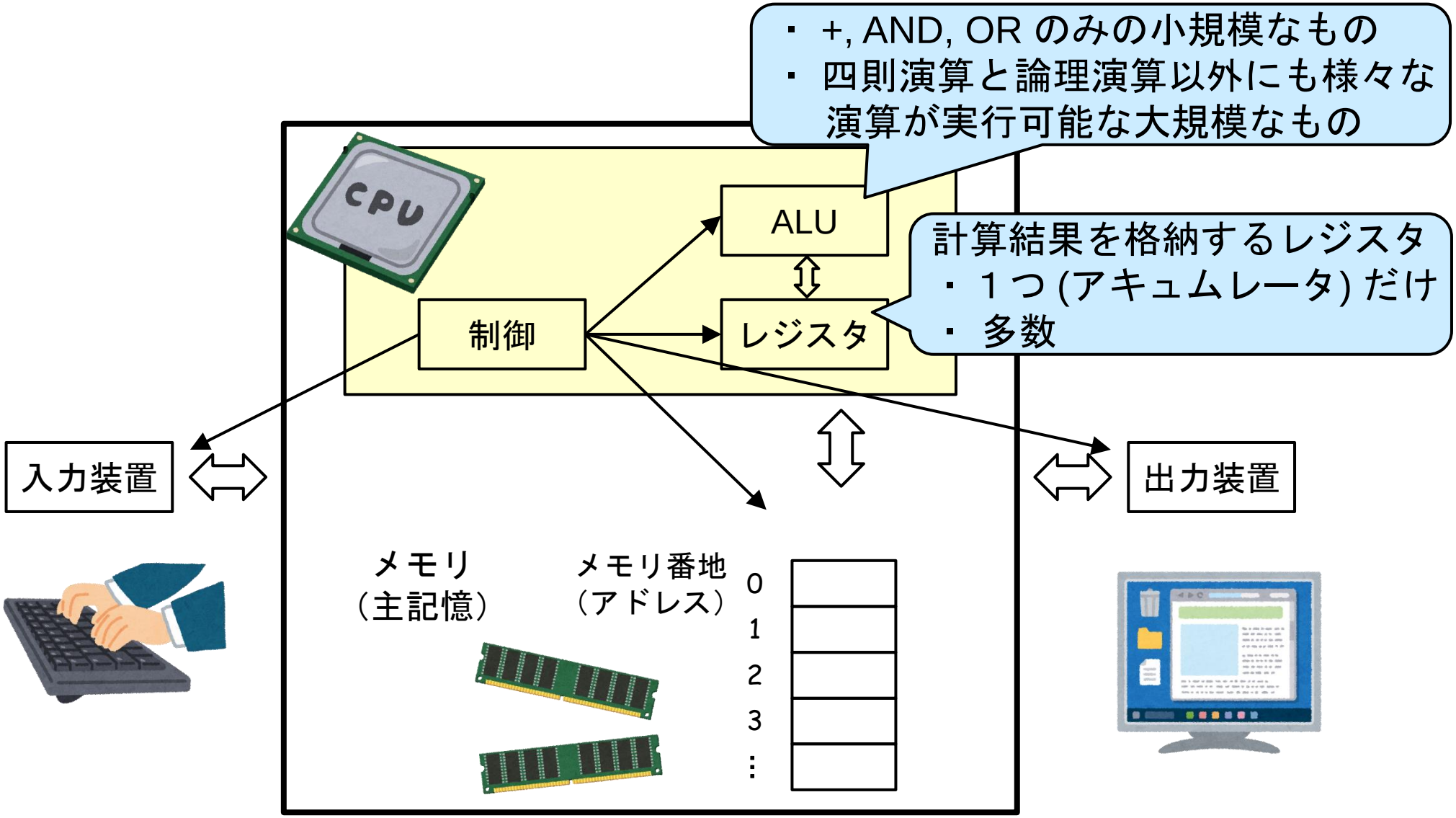
機械語命令と内部動作

- 基本的・典型的なアーキテクチャの構成
- 一般的な機械語命令の形式
- 機械語命令に対するレジスタ転送レベルの内部動作
  - フェッチ動作
  - ロード命令
  - 加算命令
  - 減算命令(→ 補数表現)
  - ストア命令
  - 条件分岐命令
  - 停止命令と割り込み動作

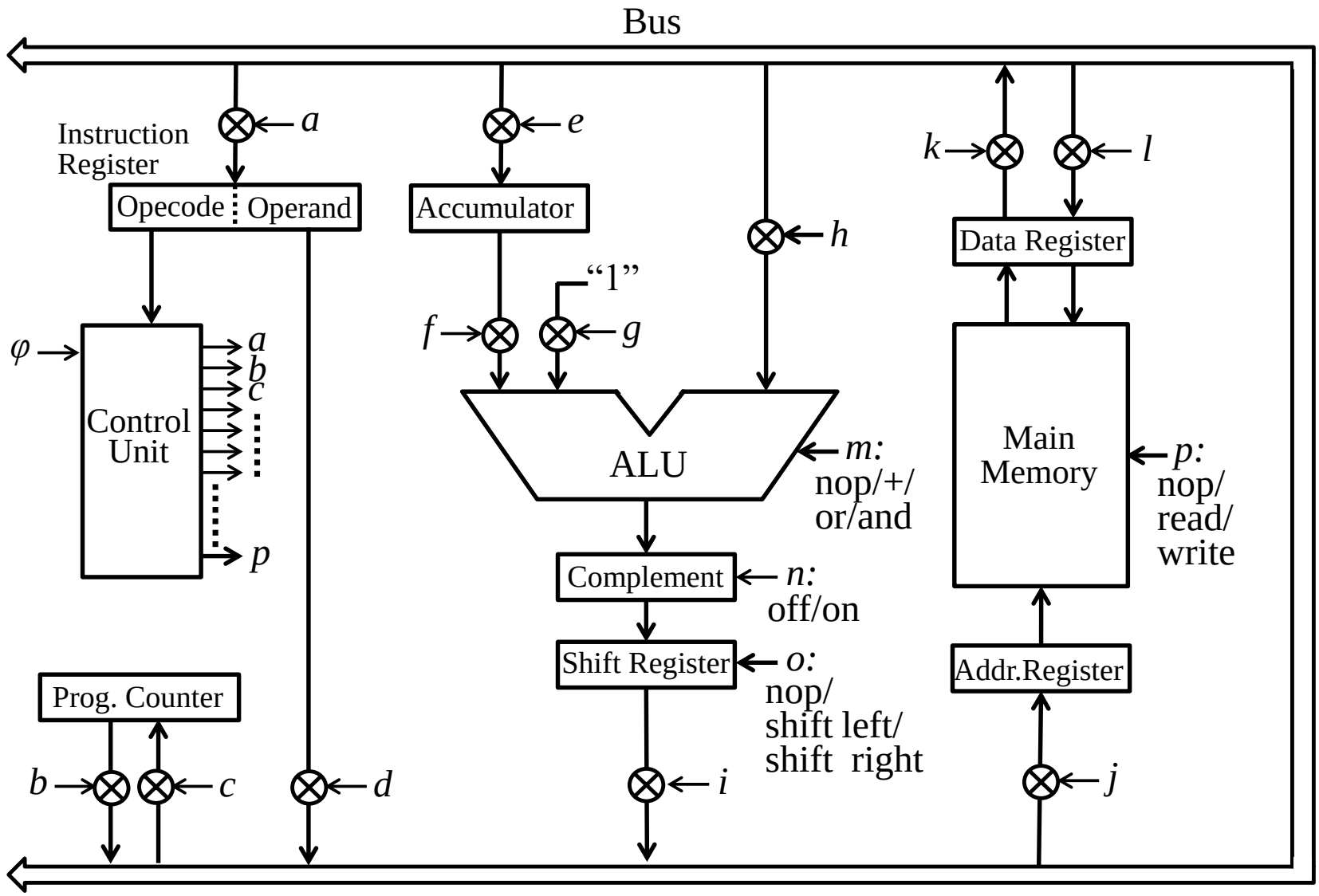
コンピュータの内部構成



コンピュータの内部構成

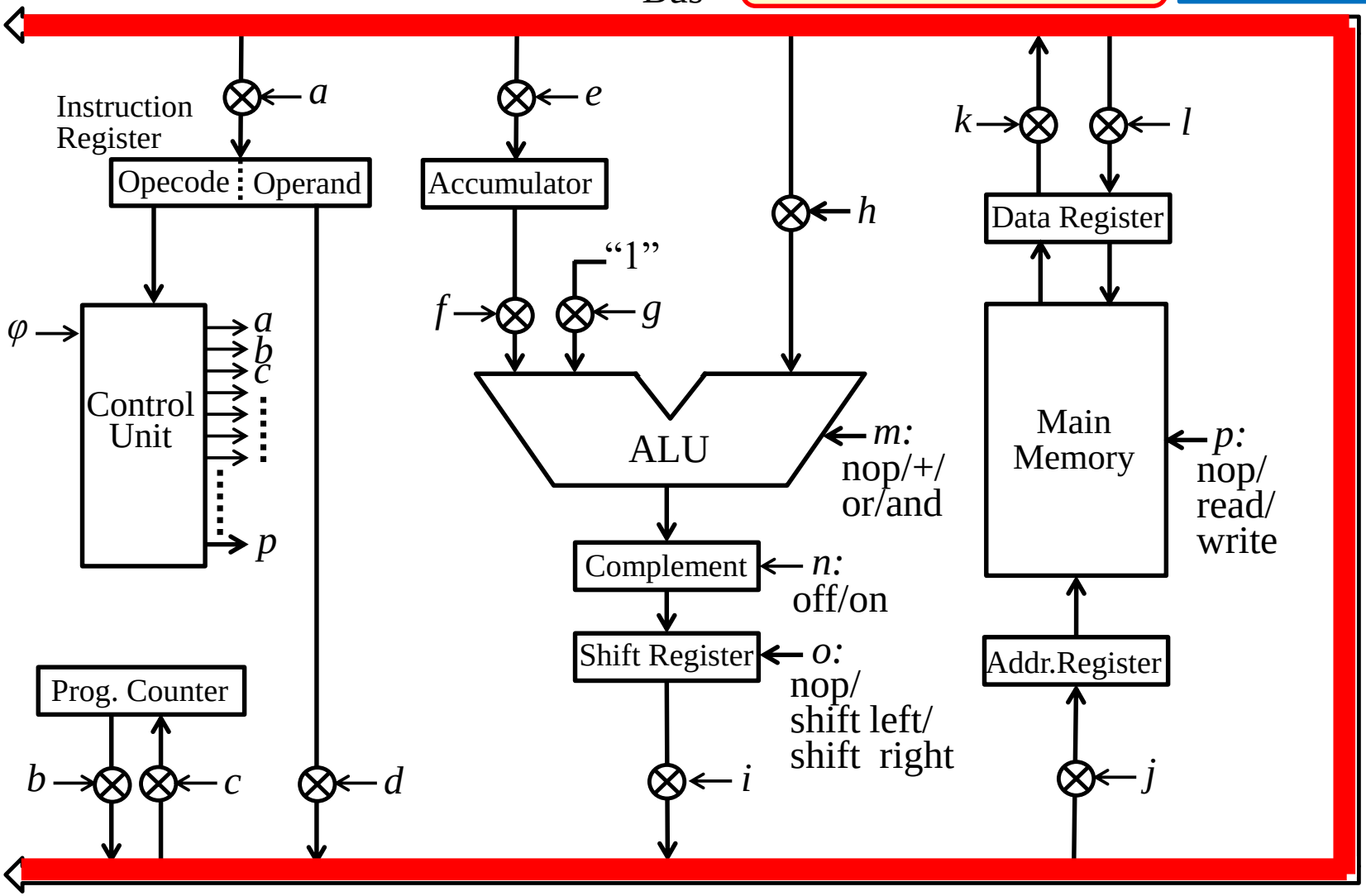


基本的・典型的なアーキテクチャの構成

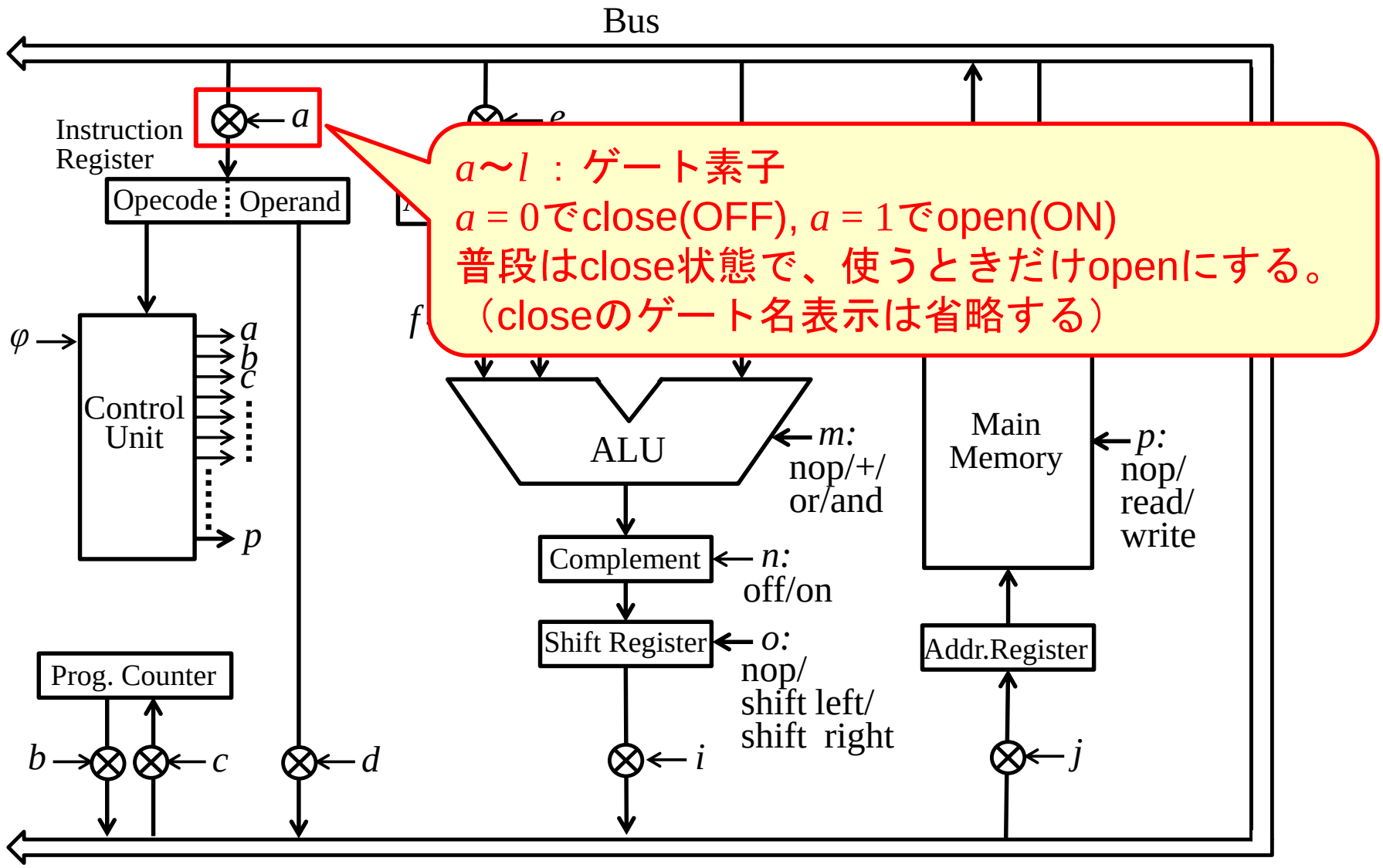


基本的・典型的なアーキテクチャの構成

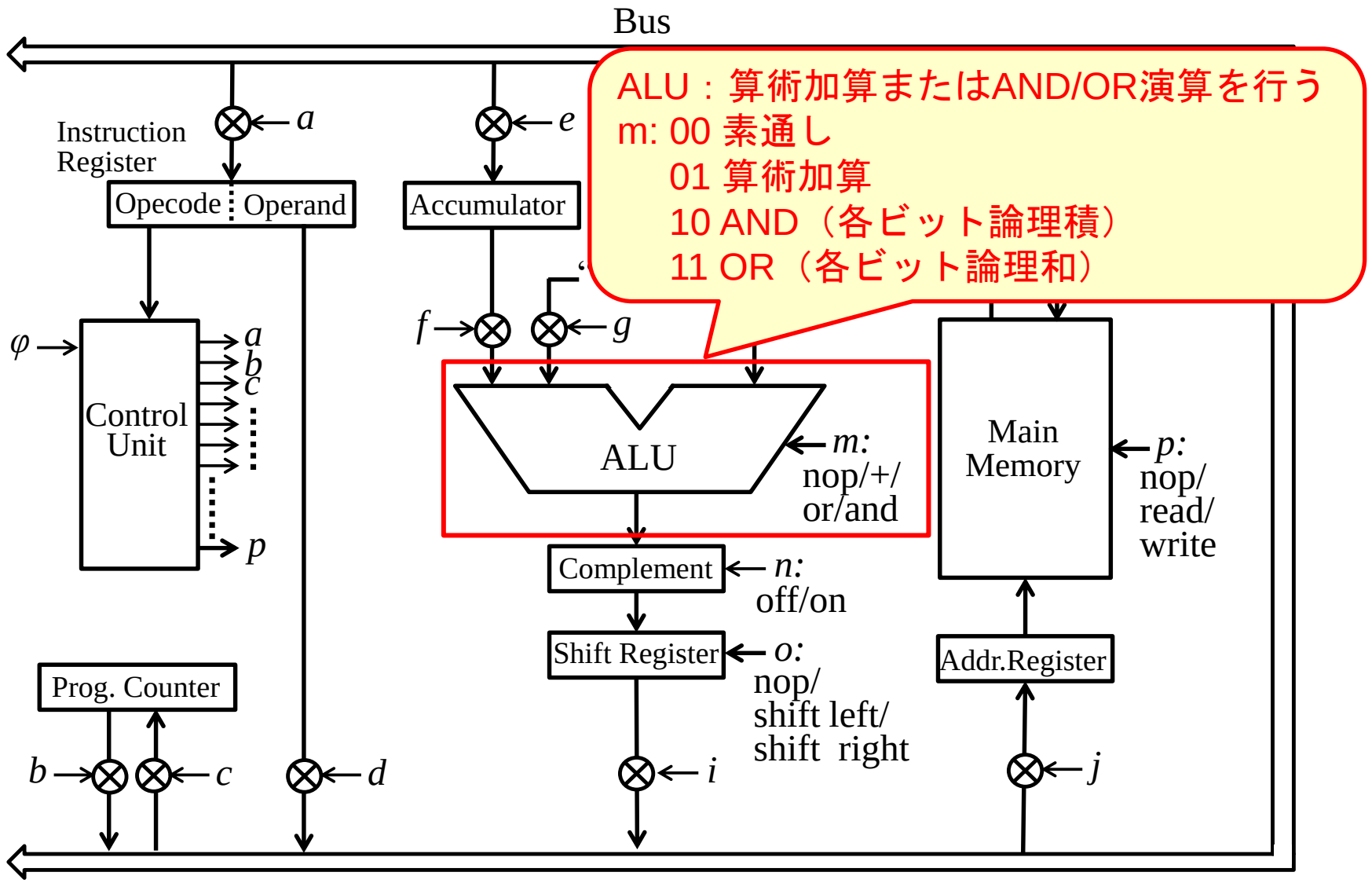
補足）ゲートを開けて，バスとデータをやり取り



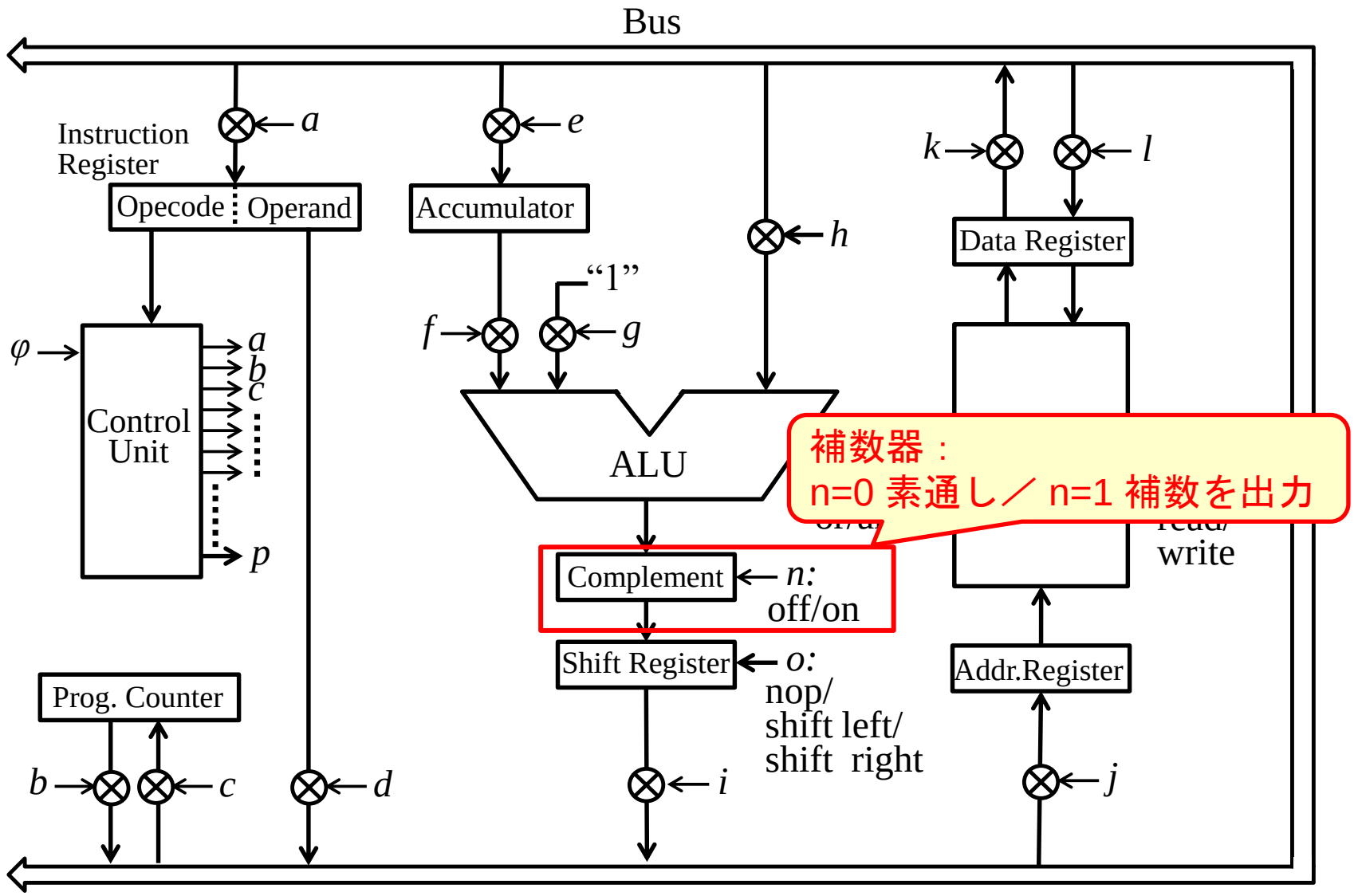
基本的・典型的なアーキテクチャの構成



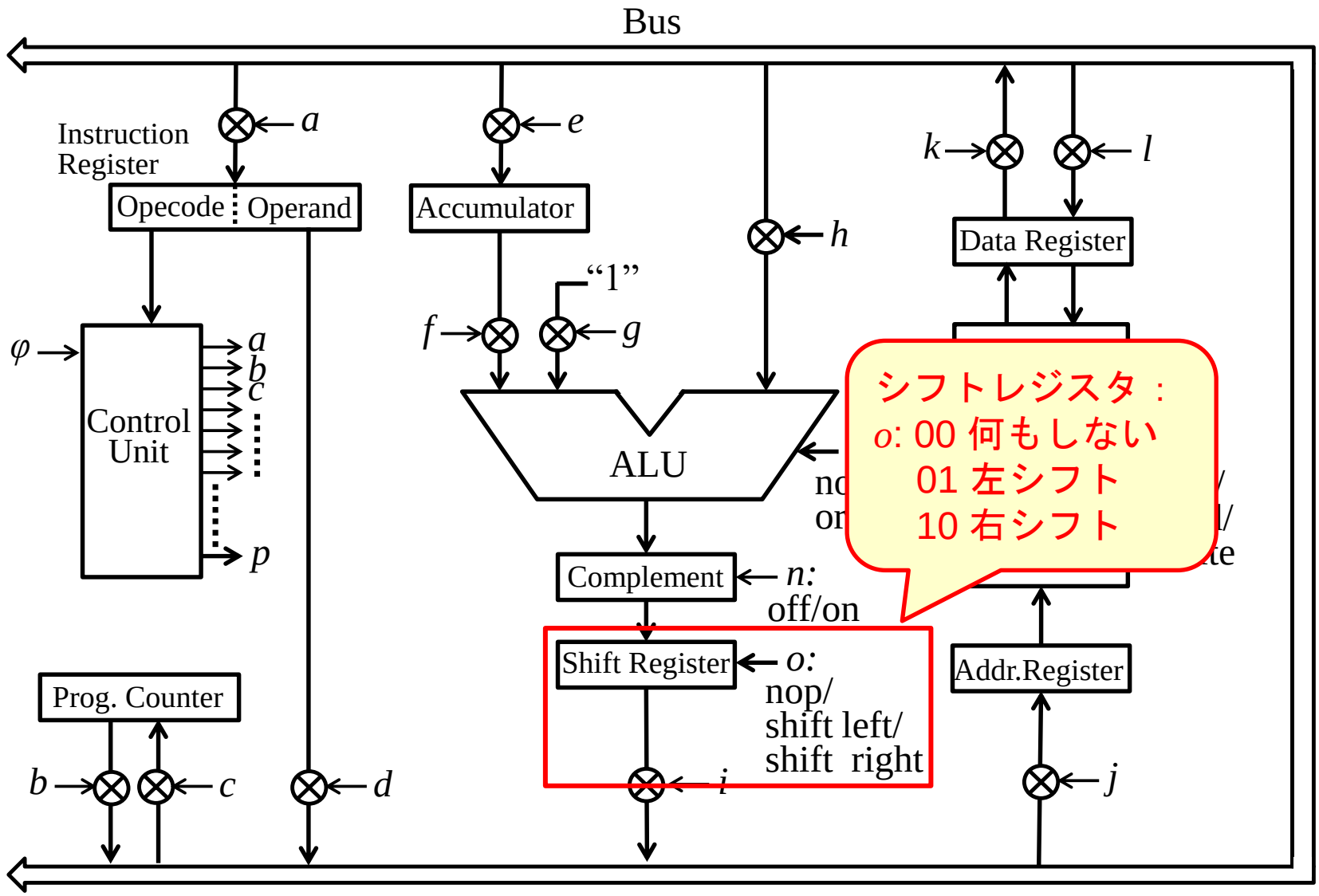
基本的・典型的なアーキテクチャの構成



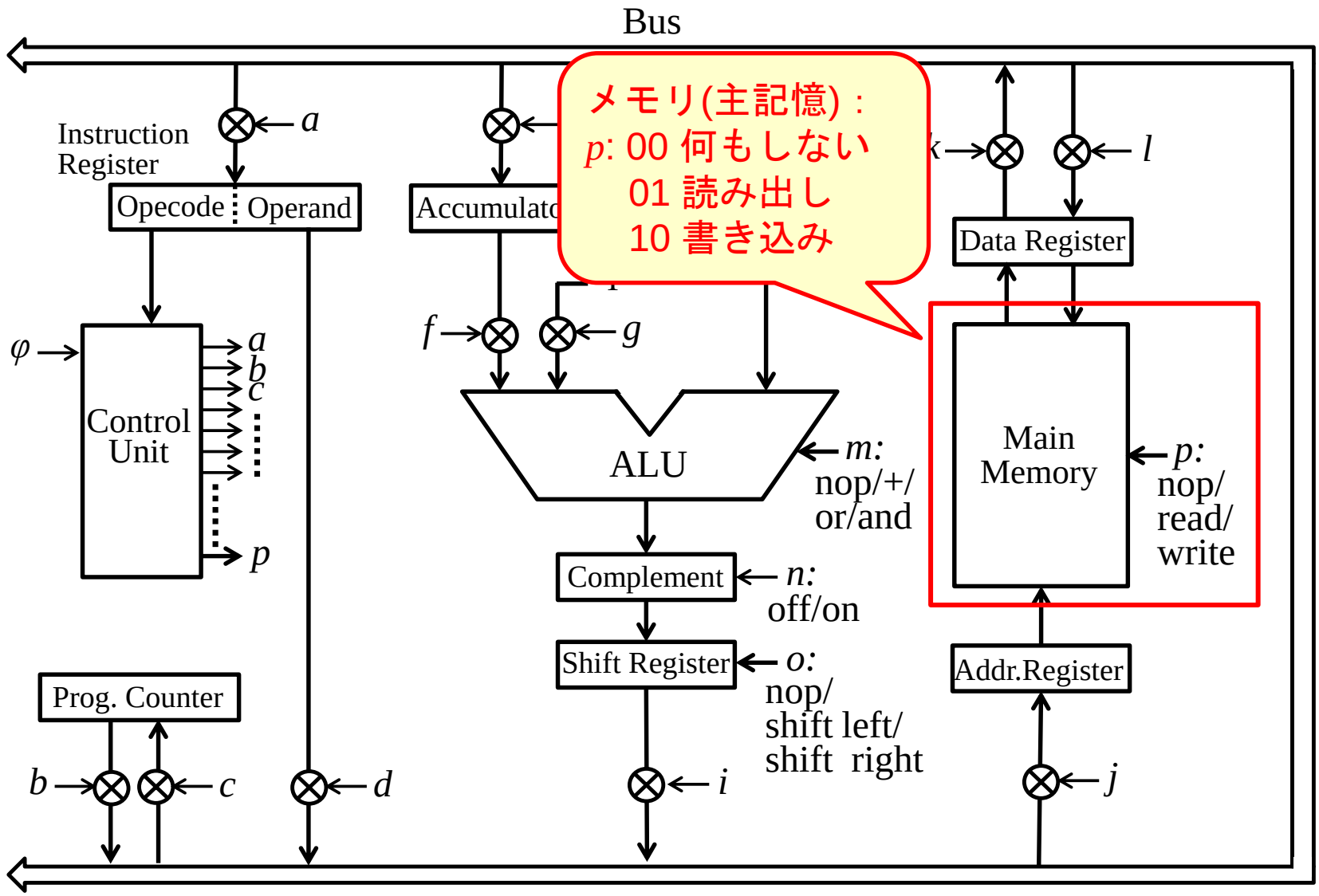
基本的・典型的なアーキテクチャの構成



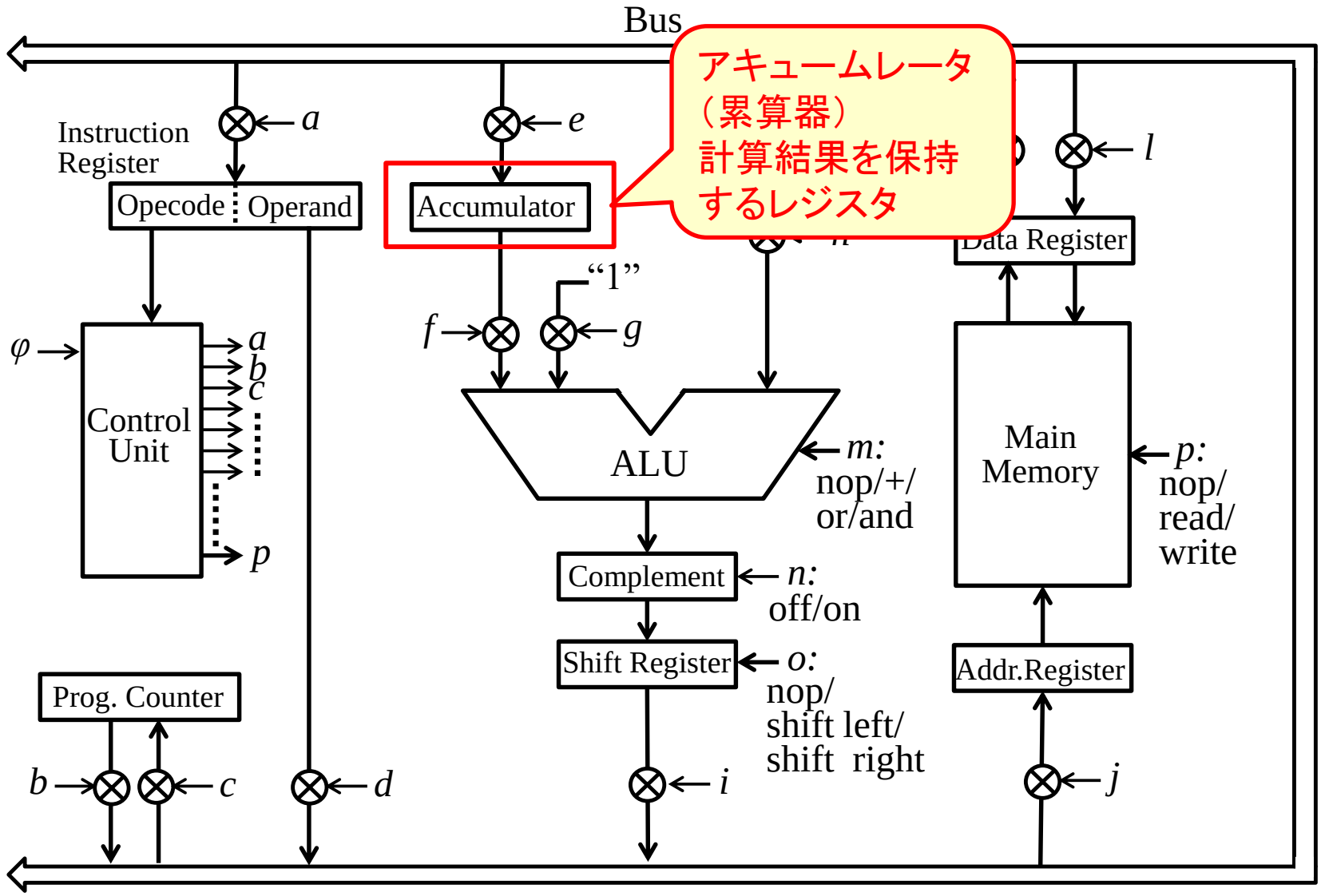
基本的・典型的なアーキテクチャの構成



基本的・典型的なアーキテクチャの構成

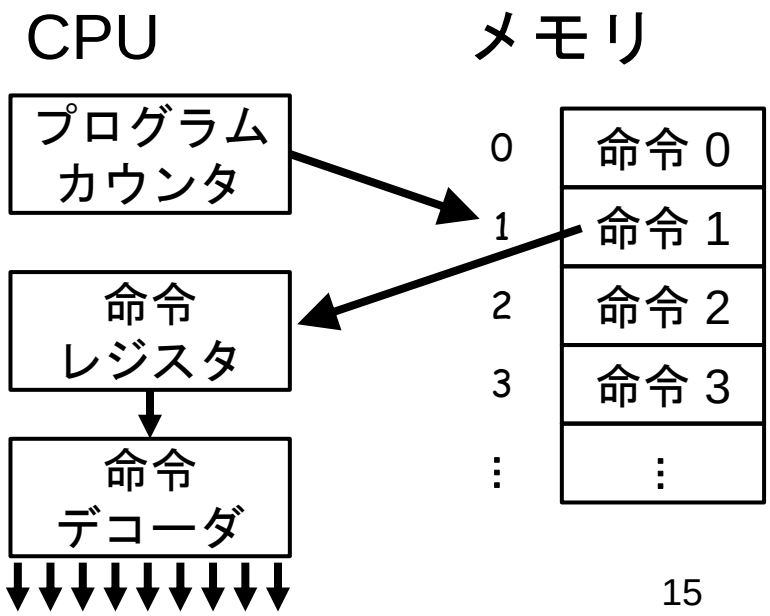


基本的・典型的なアーキテクチャの構成

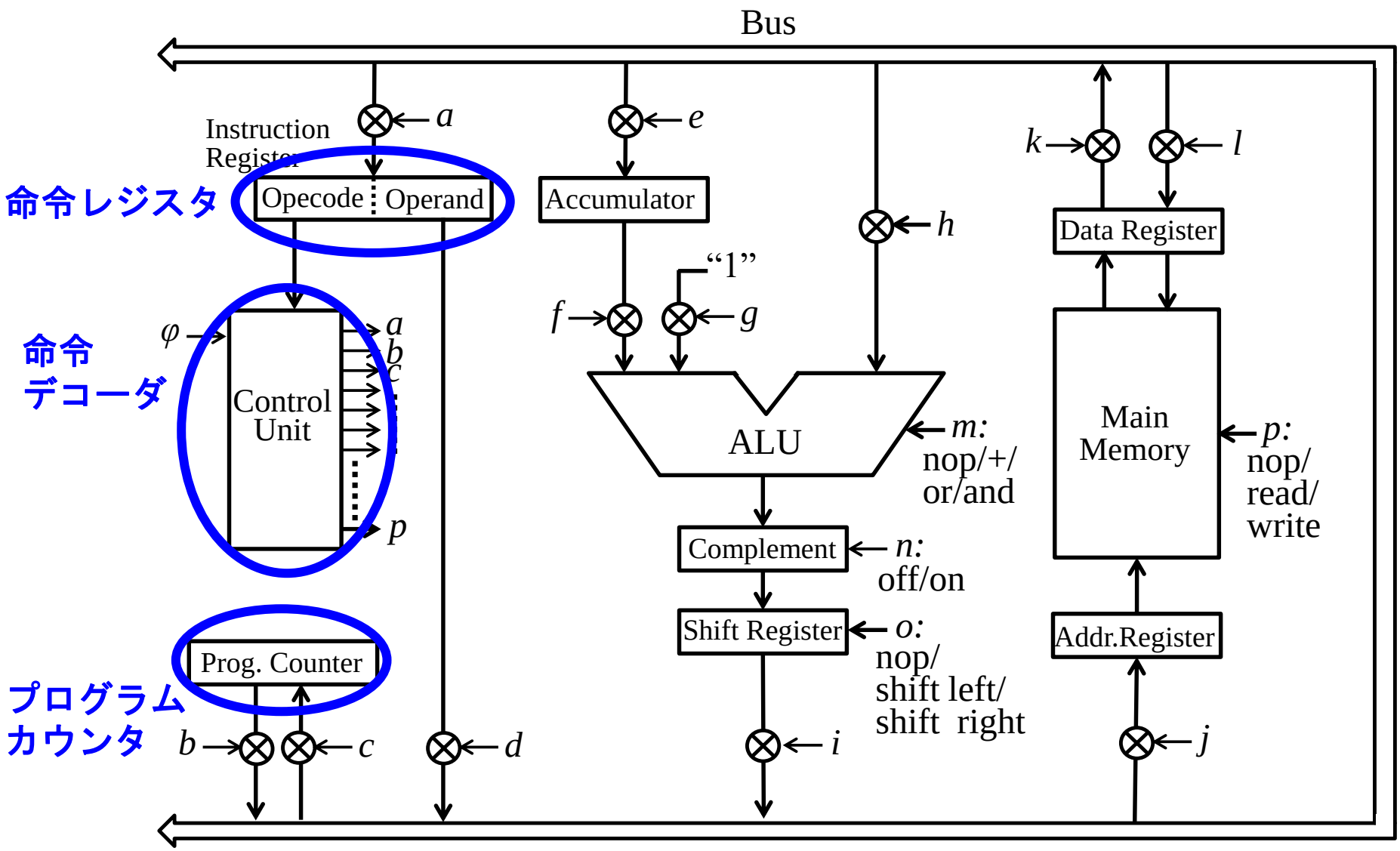


プログラムの実行 (復習)

- 機械語命令を、前から順に実行する
- **プログラム カウンタ**
  - 現在実行中の機械語命令のメモリ番地を保持する
- **命令レジスタ**
  - プログラム カウンタが指すメモリ番地の機械語命令を保持する
- **命令デコーダ**
  - 命令レジスタの命令に応じて制御信号を発生させる

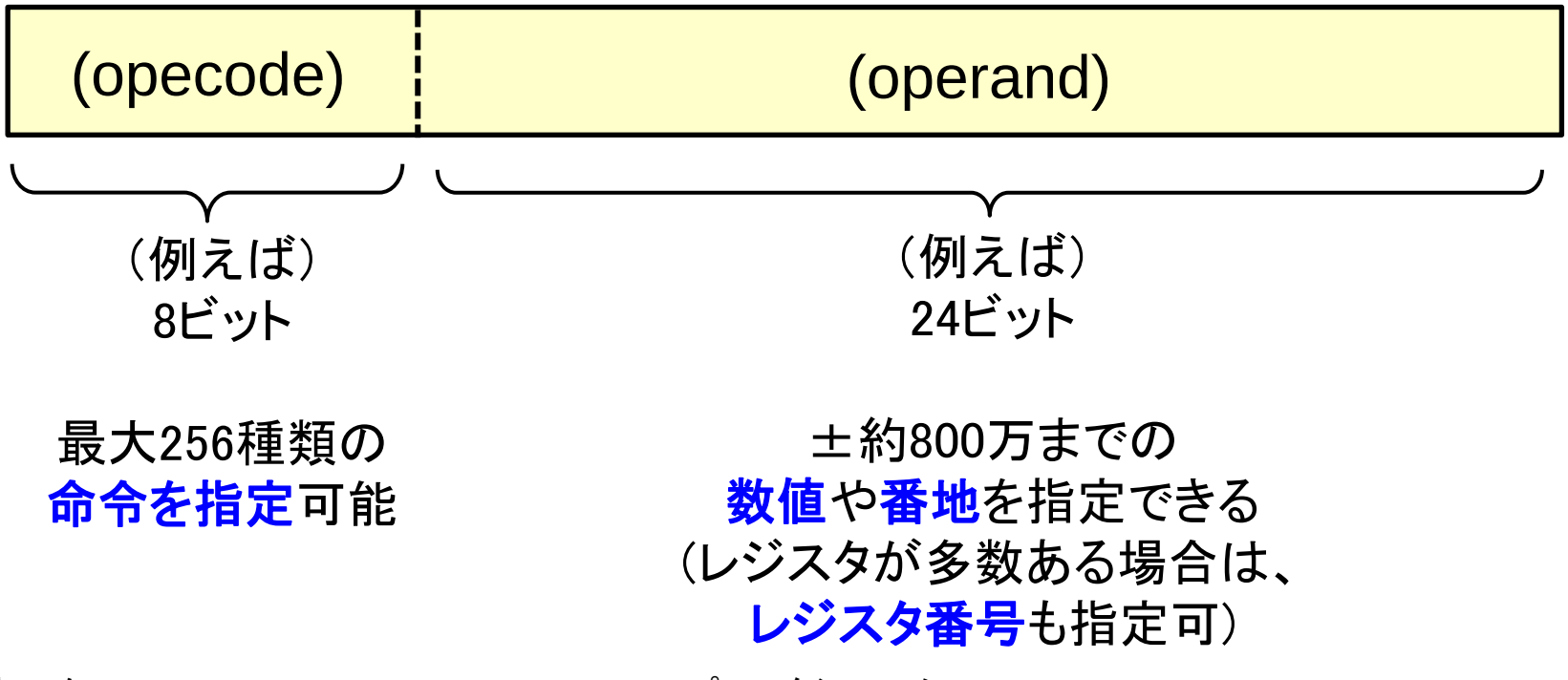


基本的・典型的なアーキテクチャの構成



典型的な機械語の構成

- オペコード (opecode) と オペランド (operand) からなる。
  - opecode: operation-codeの略。命令の種類を表す符号。
  - operand: 被演算子。数値や番地などを表す2進数。



機械語命令の実行

- 機械語１命令を、さらに細かい何段階かの動作に分解し、クロックごとに順に実行することが多い

(何段に分けるか、各段階でどの回路を動かすかは設計による)

実行の各段階

- フェッチ (Fetch)
  - 機械語命令をメモリから読んで、命令レジスタに格納する
- デコード (Decode)
  - 命令レジスタの命令を読んで、それに応じた制御をする
- 実行 (Execute)
  - 命令を実行する

休 憩

- ここで、少し休憩しましょう。

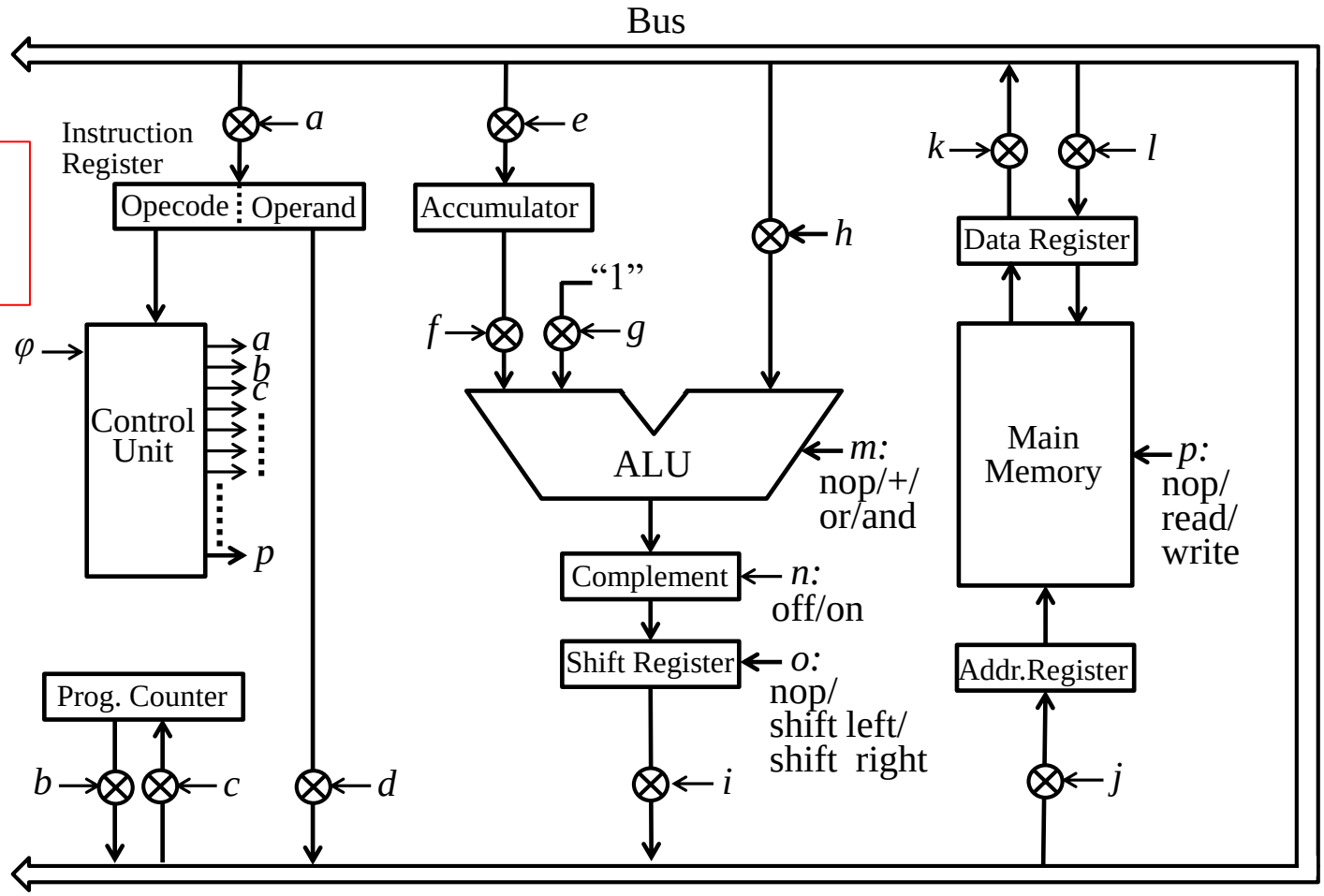
- 深呼吸したり、肩の力を抜いてから、
  次のビデオに進んでください。

フェッチ(Fetch)動作

- プログラムカウンタが指すメモリ番地の機械語命令を読み出して命令レジスタに格納する基本動作

clock0: *b*, *j*, *p*←"read"
clock1: *k*, *a*

ほぼすべての命令に含まれる基本的な部分動作

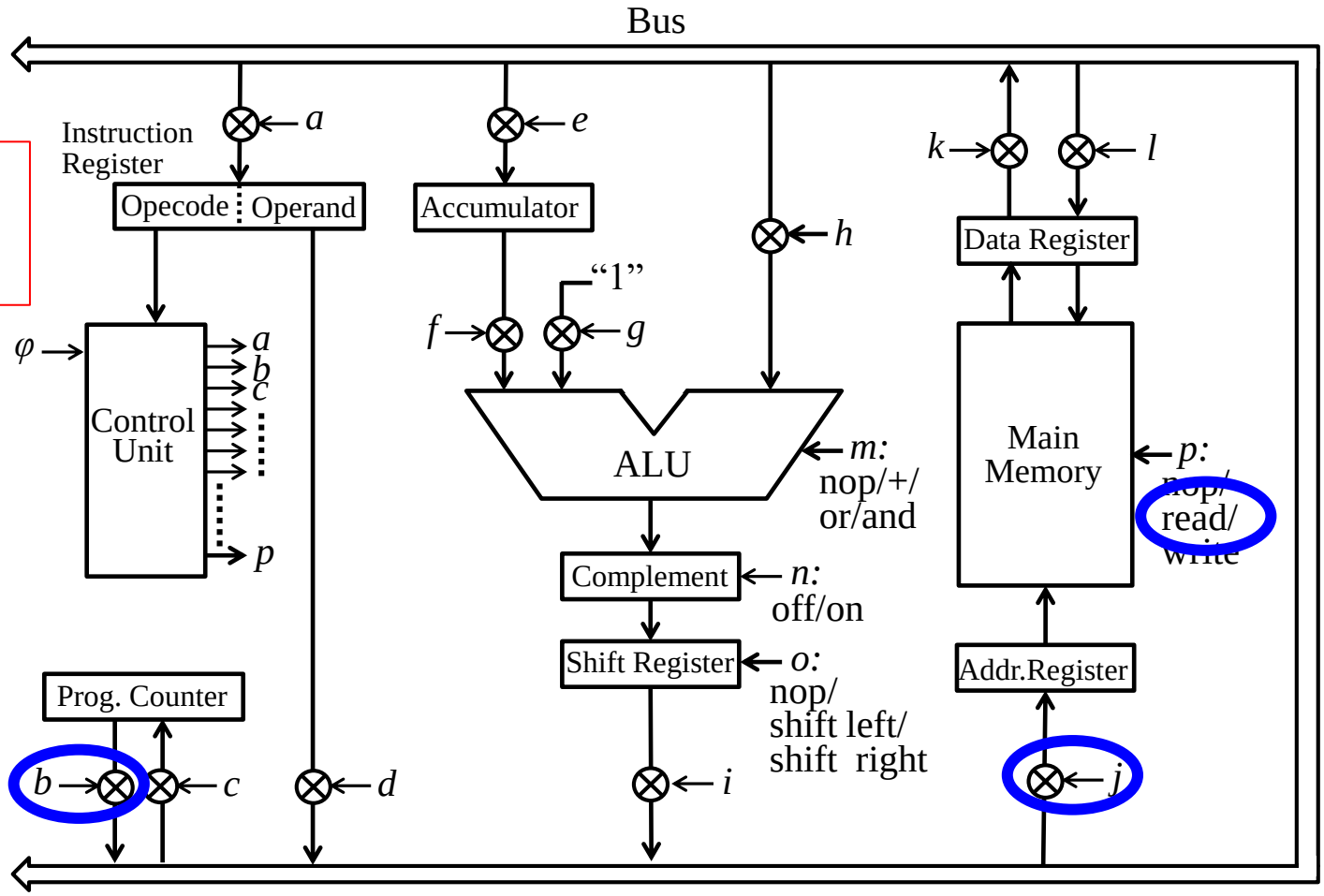


フェッチ(Fetch)動作

- プログラムカウンタが指すメモリ番地の機械語命令を読み出して命令レジスタに格納する基本動作

clock0: *b*, *j*, *p*←"read"
clock1: *k*, *a*

ほぼすべての命令に含まれる基本的な部分動作



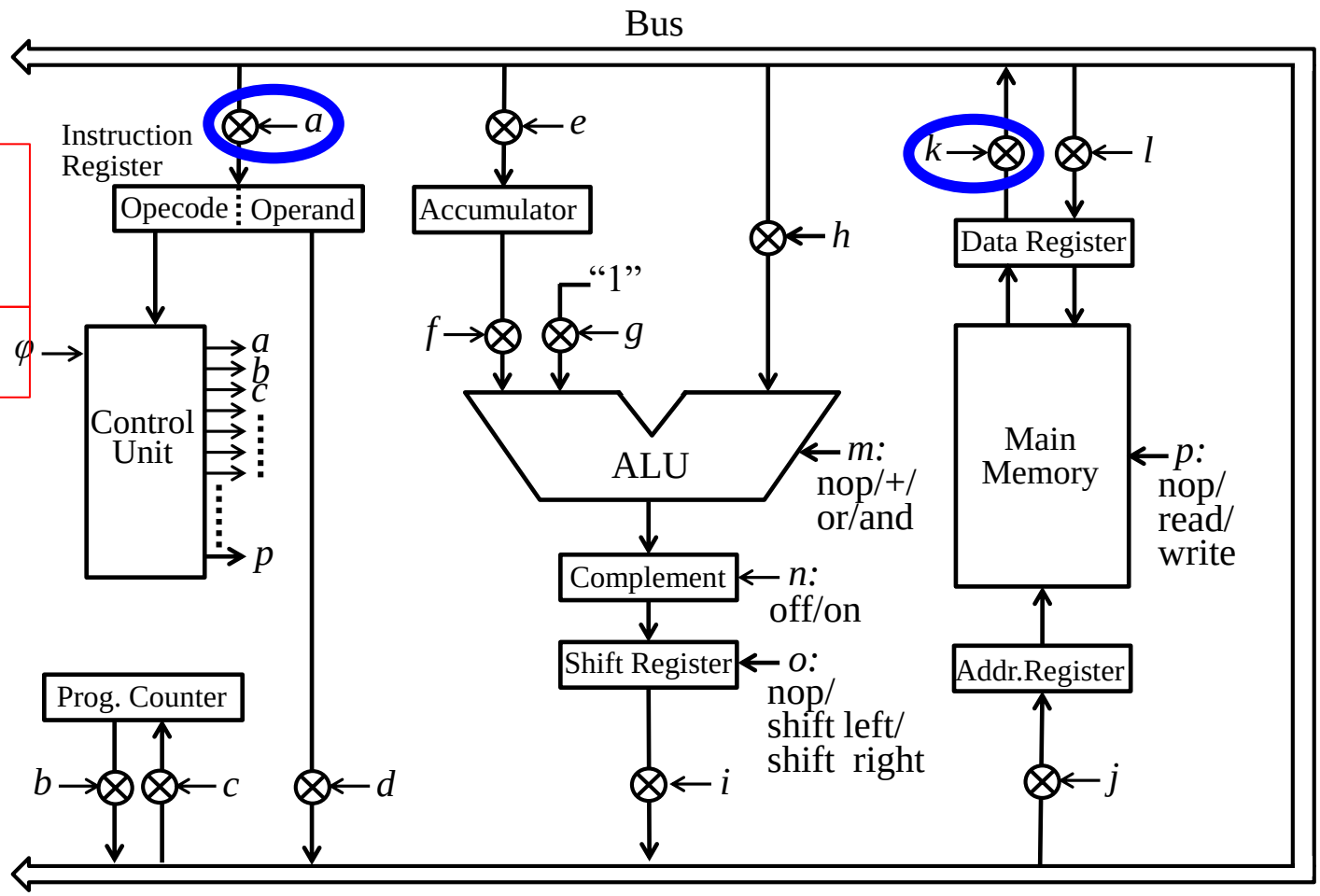
フェッチ(Fetch)動作

- プログラムカウンタが指すメモリ番地の機械語命令を読み出して命令レジスタに格納する基本動作

clock0: *b*, *j*, *p*←"read"
clock1: *k*, *a*

clock2: デコード

ほぼすべての命令に含まれる基本的な部分動作

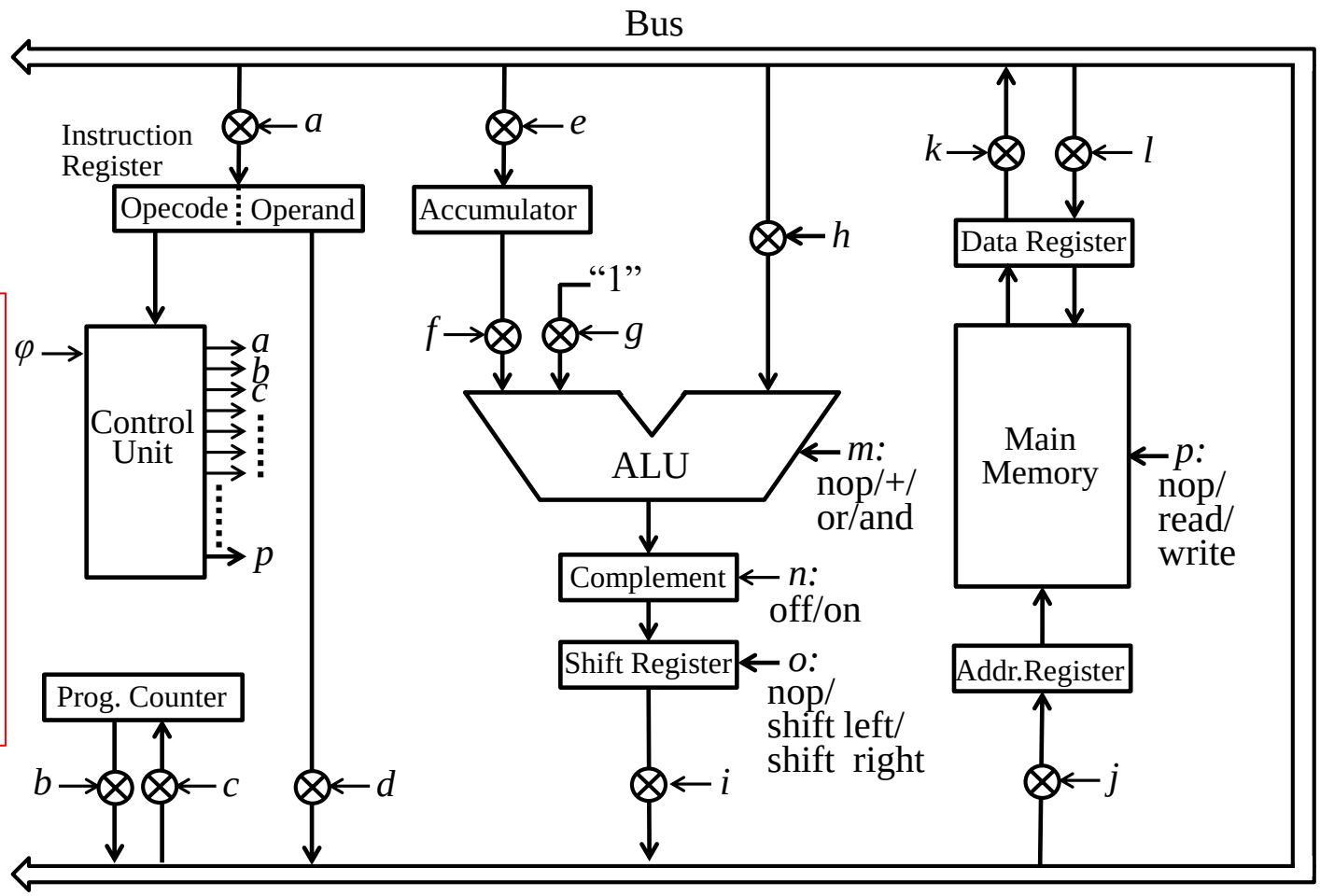


ロード(Load)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に格納する

100番地の内容を
Accにセットする

L 100
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *e*
clock5: *b*, *h*, *g*, *m*←"+"
clock6: *i*, *c*
（次のフェッチ動作へ）

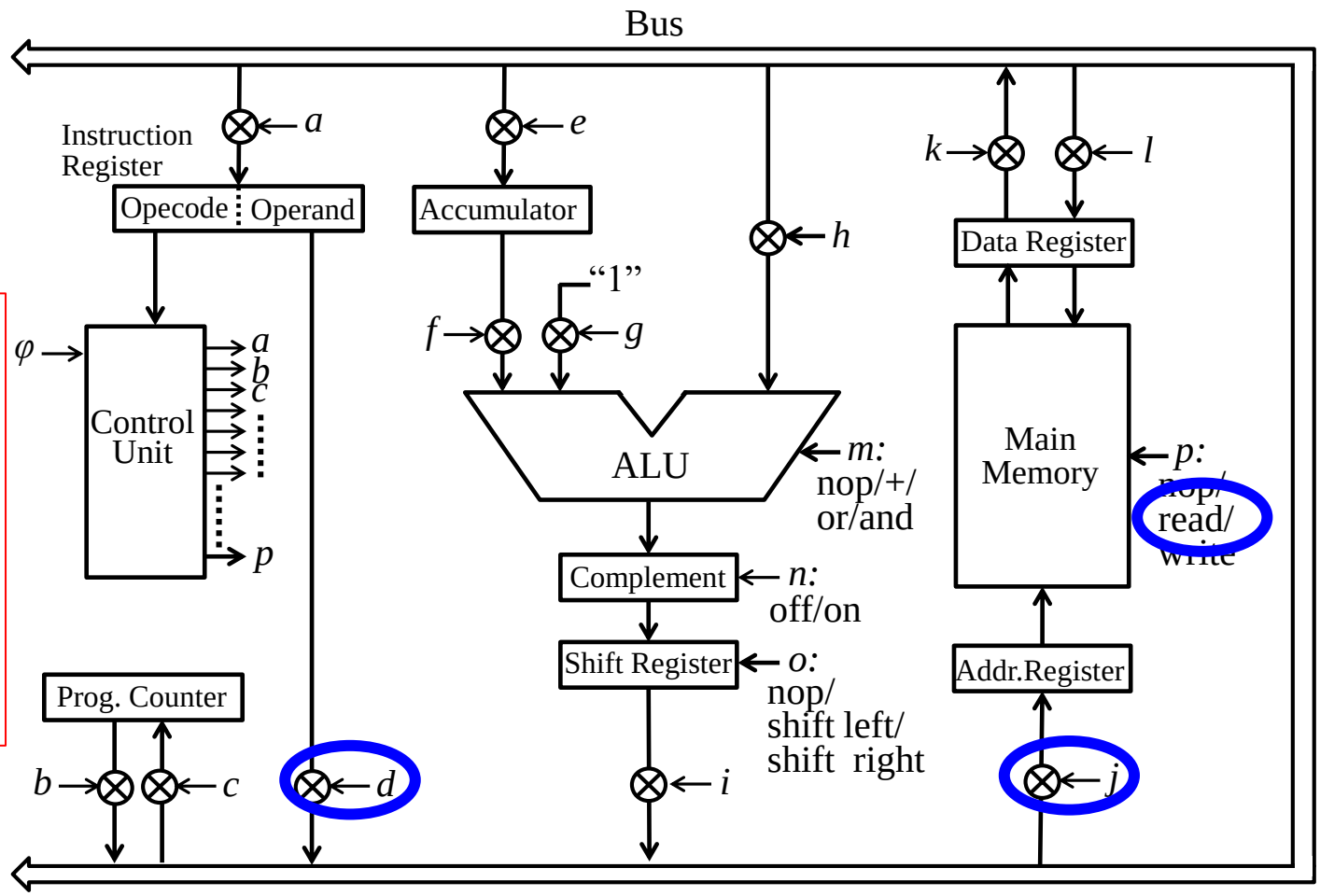


ロード(Load)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に格納する

100番地の内容を
Accにセットする

L 100
clock3: *d*, *j*, *p*←“read”
clock4: *k*, *e*
clock5: *b*, *h*, *g*, *m*←“+”
clock6: *i*, *c*
（次のフェッチ動作へ）

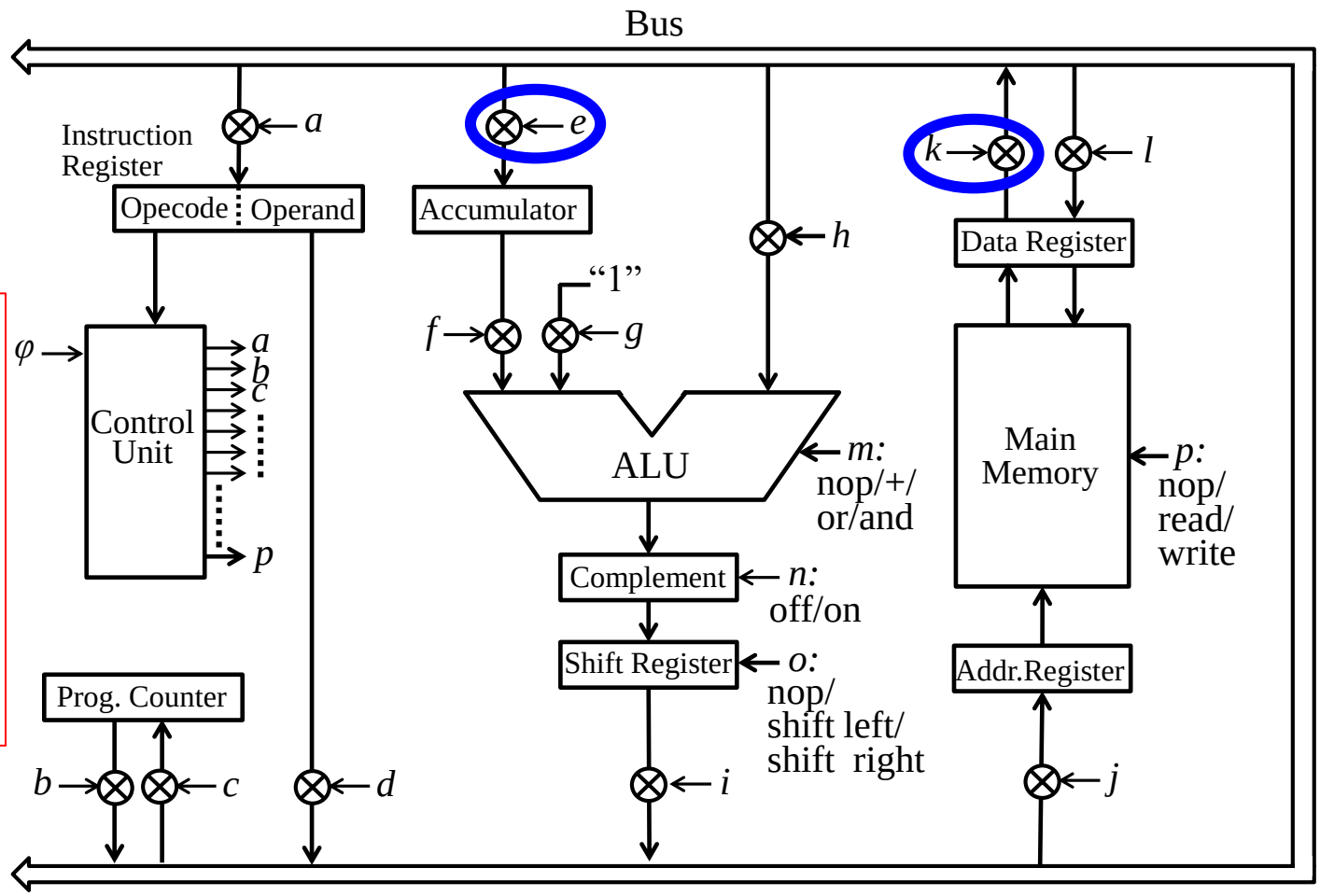


ロード(Load)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に格納する

100番地の内容を
Accにセットする

L 100
clock3: *d*, *j*, *p*←“read”
clock4: *k*, *e*
clock5: *b*, *h*, *g*, *m*←“+”
clock6: *i*, *c*
（次のフェッチ動作へ）

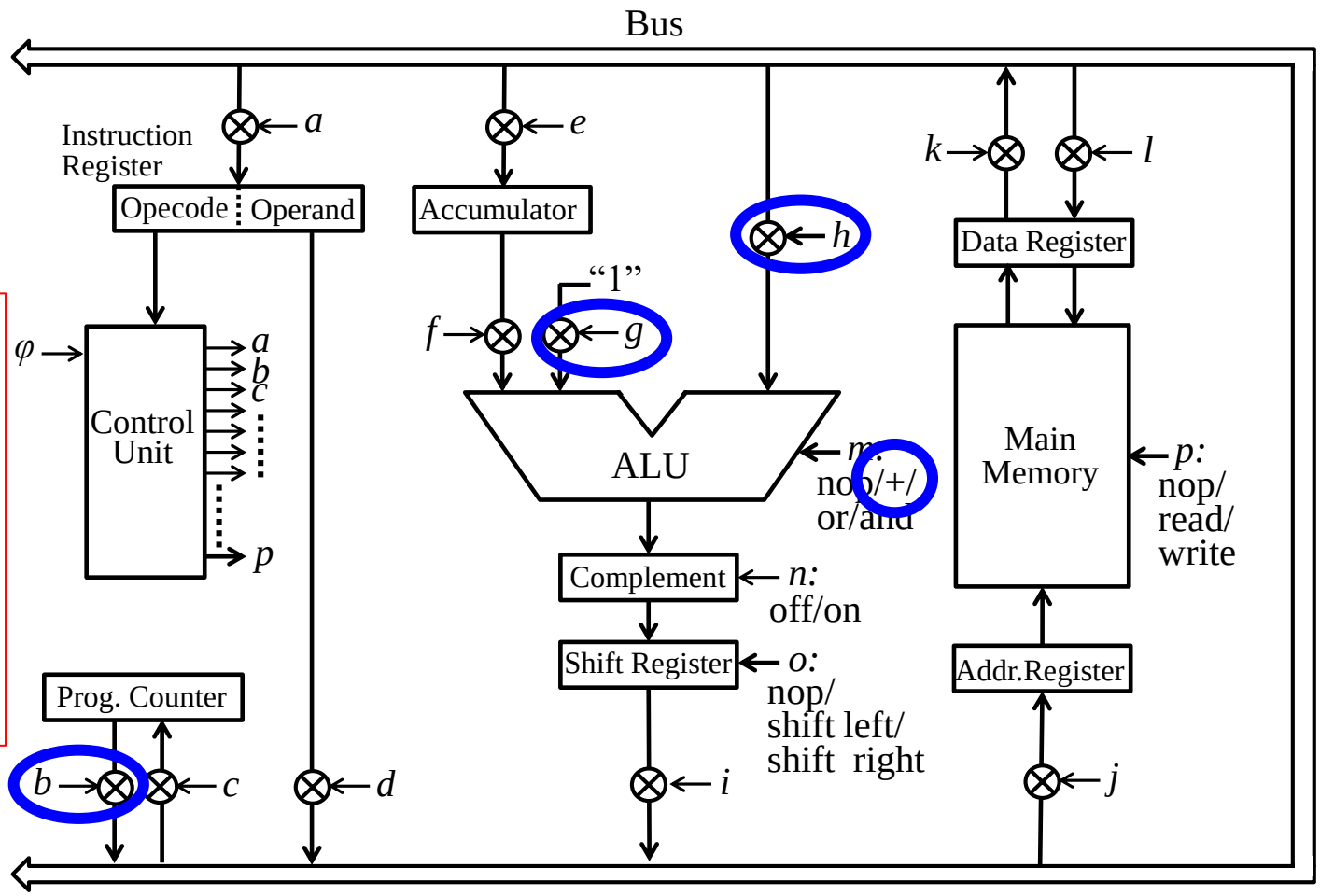


ロード(Load)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に格納する

100番地の内容を
Accにセットする

L 100
clock3: *d*, *j*, *p*←“read”
clock4: *k*, *e*
clock5: *b*, *h*, *g*, *m*←“+”
clock6: *i*, *c*
（次のフェッチ動作へ）



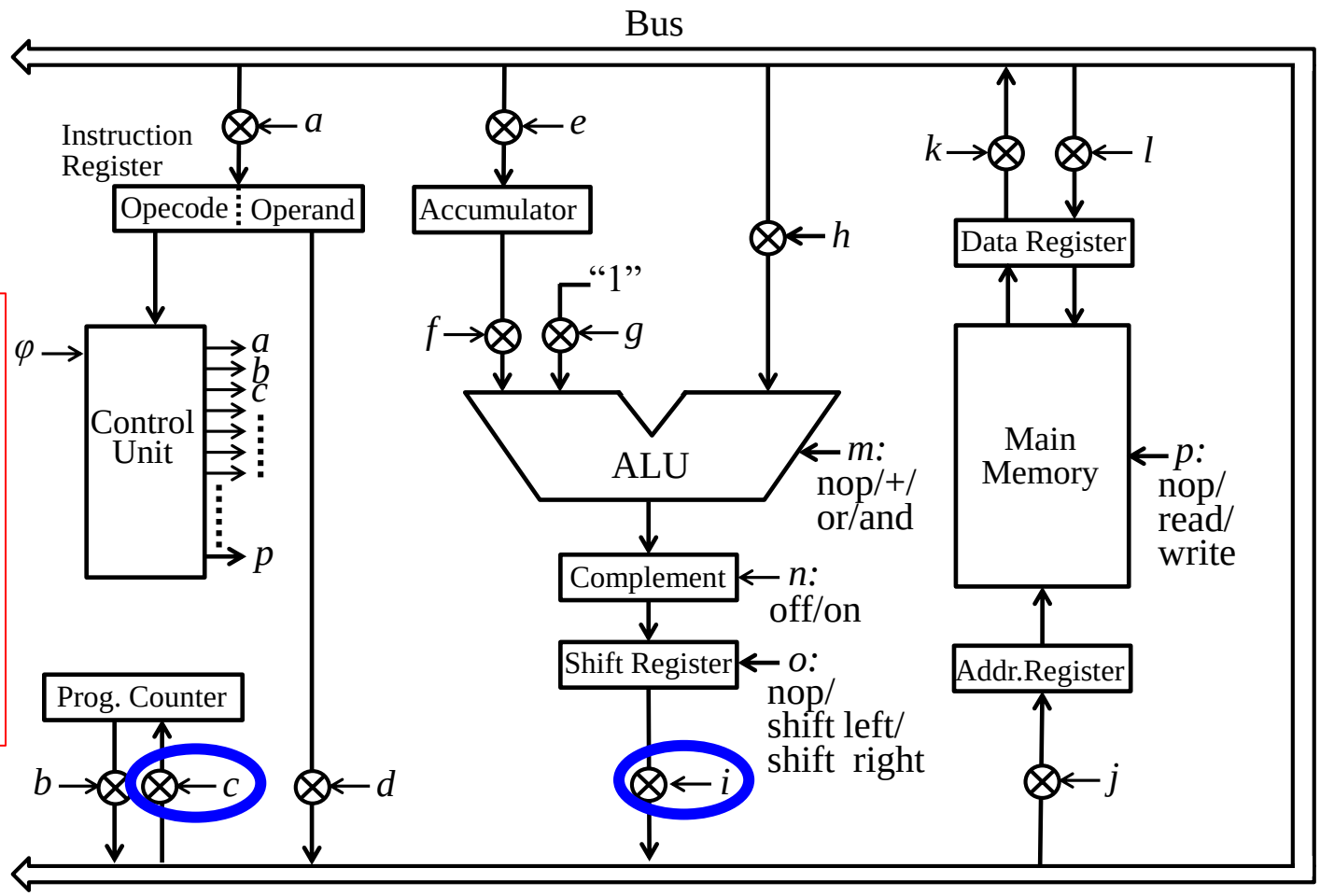
Clock5, 6:
プログラムカウンタ + 1

ロード(Load)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に格納する

100番地の内容を
Accにセットする

L 100
clock3: *d*, *j*, *p*←“read”
clock4: *k*, *e*
clock5: *b*, *h*, *g*, *m*←“+”
clock6: *i*, *c*
（次のフェッチ動作へ）



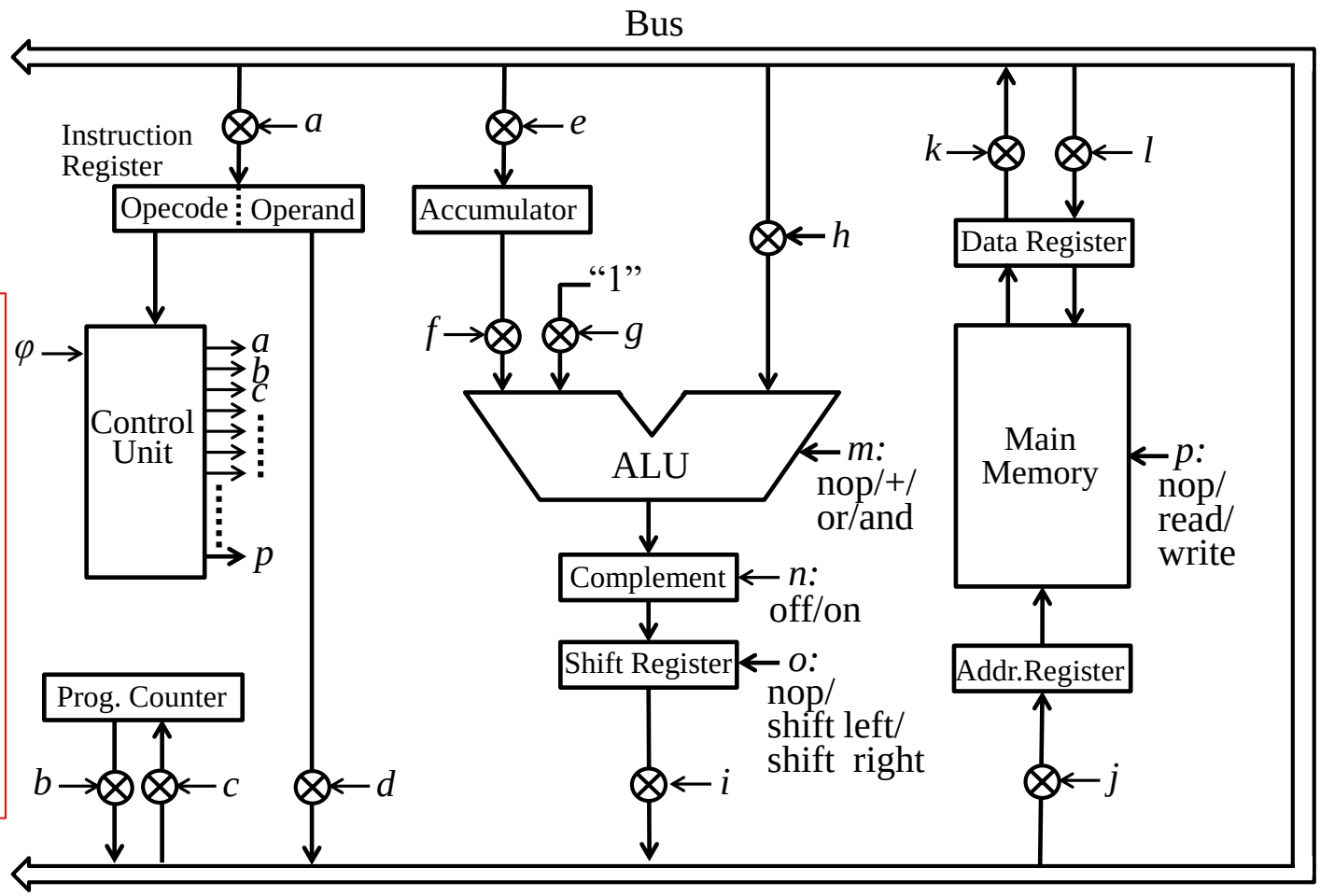
Clock5, 6:
プログラムカウンタ + 1

加算(Addition)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に加算し、結果をAccに格納する。

Accの内容と200番地の内容を加算し、結果をAccに保持

A 200
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *f*, *m*←"+"
clock5: *i*, *e*
clock6: *b*, *h*, *g*, *m*←"+"
clock7: *i*, *c*
（次のフェッチ動作へ）

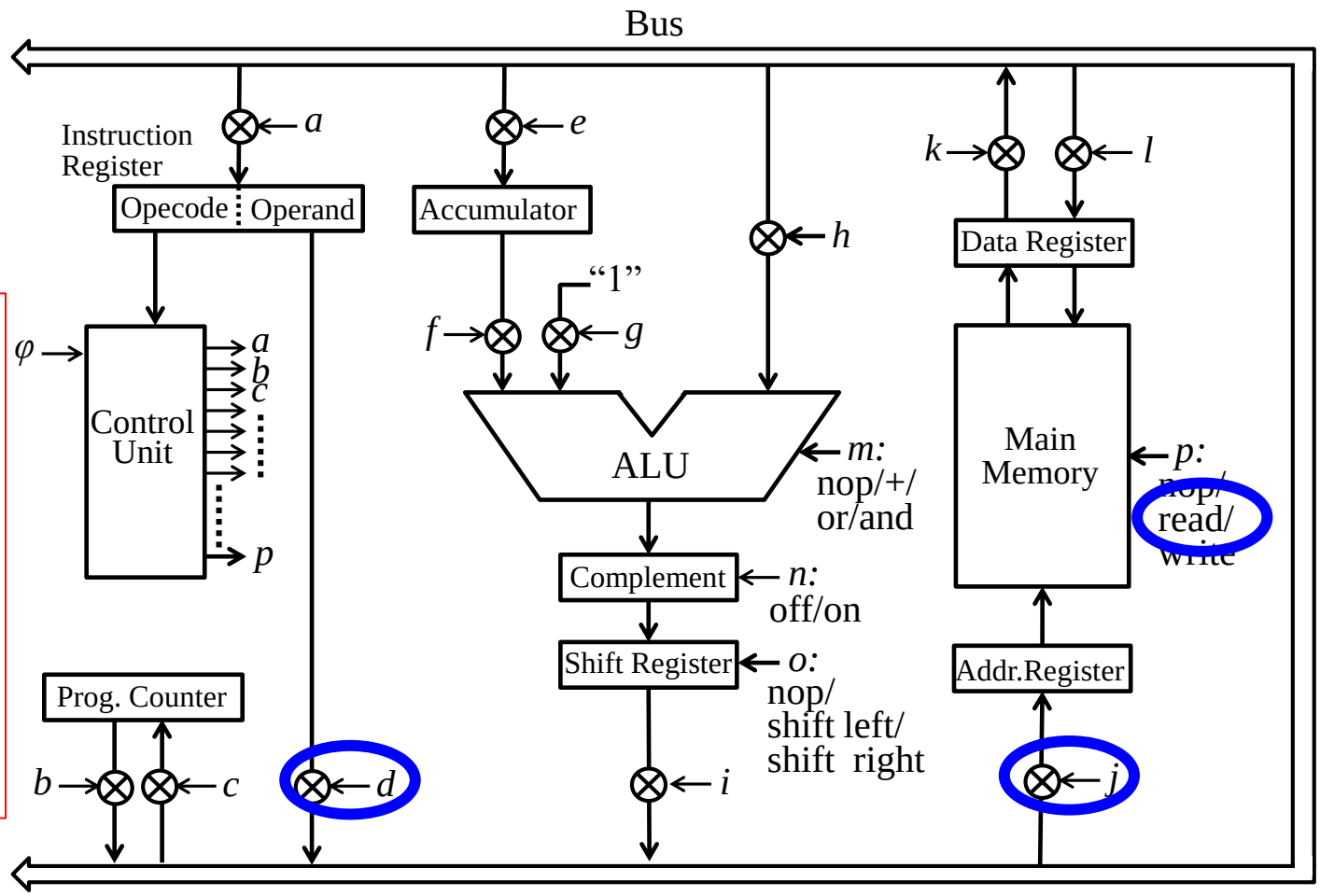


加算(Addition)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に加算し、結果をAccに格納する。

Accの内容と200番地の内容を加算し、結果をAccに保持

A 200
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *f*, *m*←"+"
clock5: *i*, *e*
clock6: *b*, *h*, *g*, *m*←"+"
clock7: *i*, *c*
（次のフェッチ動作へ）

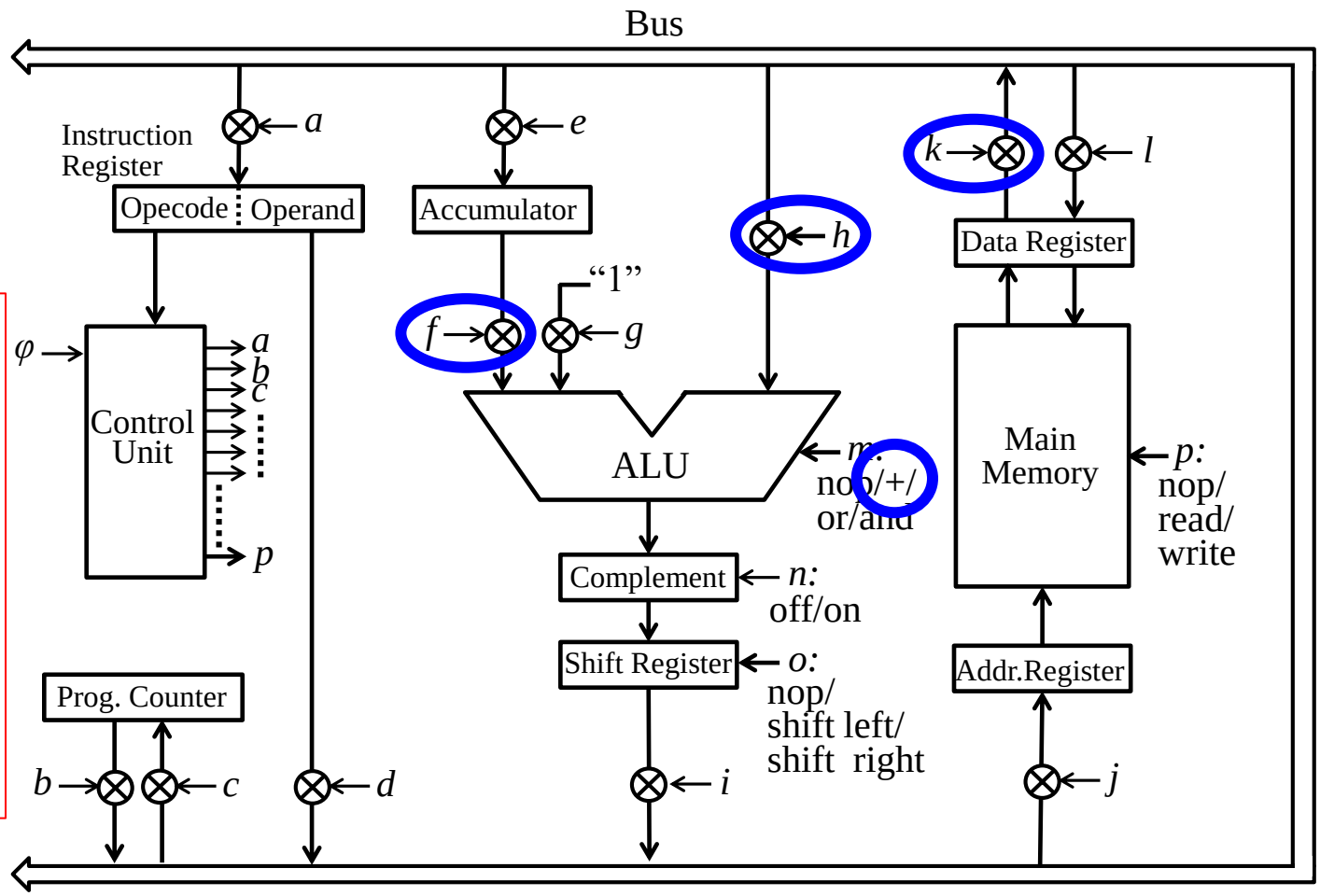


加算(Addition)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に加算し、結果をAccに格納する。

Accの内容と200番地の内容を加算し、結果をAccに保持

A 200
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *f*, *m*←"+"
clock5: *i*, *e*
clock6: *b*, *h*, *g*, *m*←"+"
clock7: *i*, *c*
（次のフェッチ動作へ）

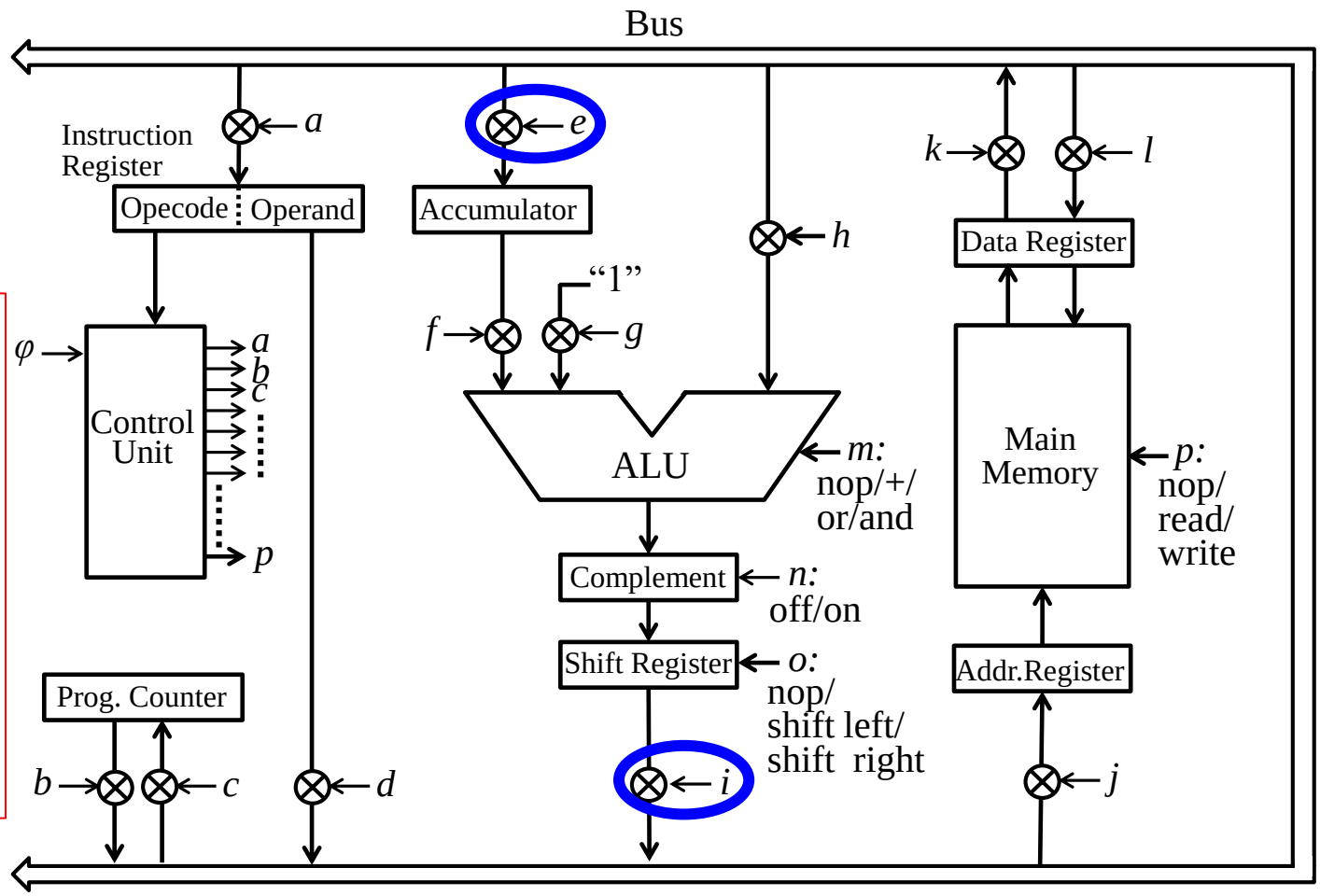


加算(Addition)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に加算し、結果をAccに格納する。

Accの内容と200番地の内容を加算し、結果をAccに保持

A 200
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *f*, *m*←"+"
clock5: *i*, *e*
clock6: *b*, *h*, *g*, *m*←"+"
clock7: *i*, *c*
（次のフェッチ動作へ）



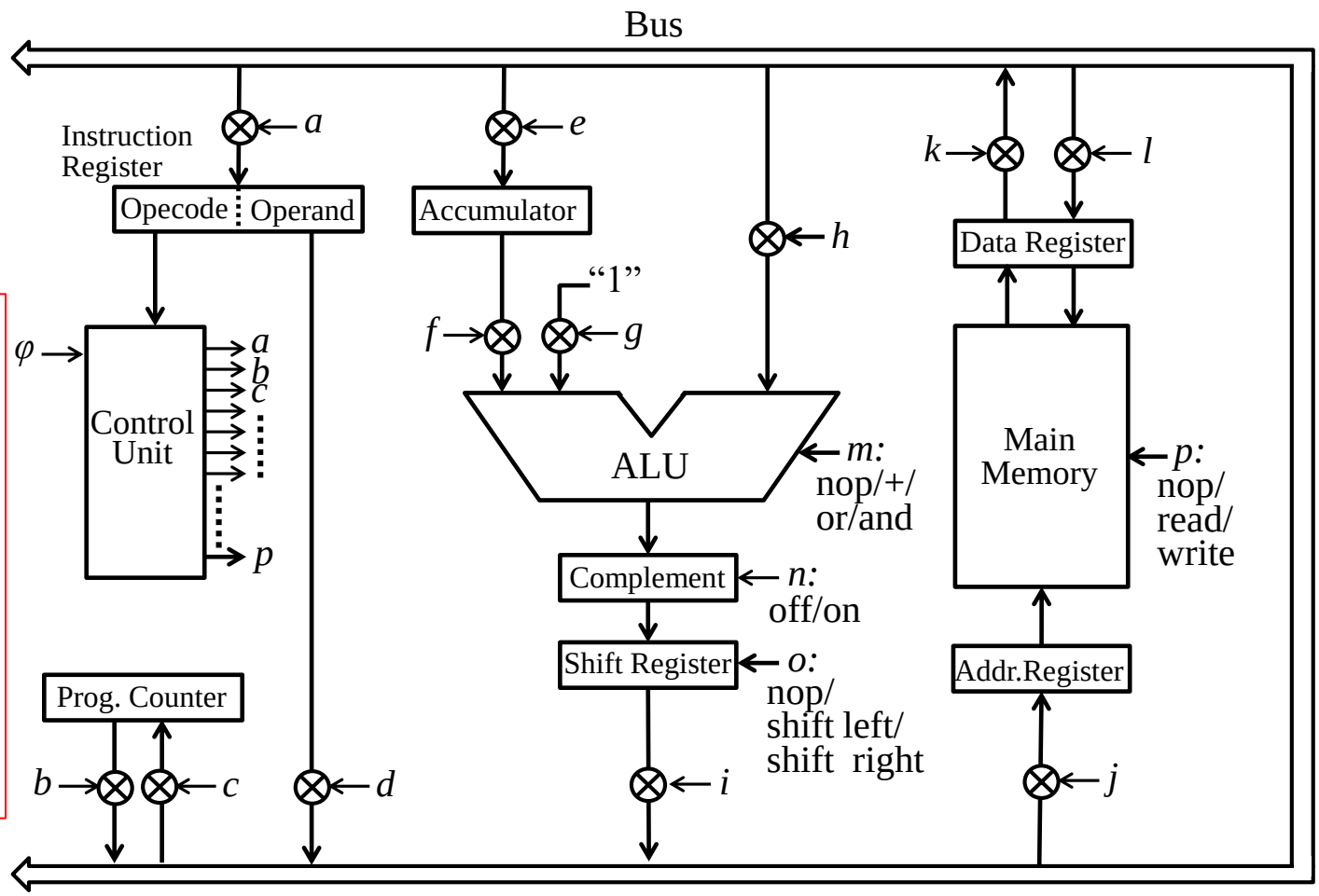
加算(Addition)命令

- オペランドで指定された番地のデータを読み出して
  アキュムレータ(Acc)に加算し、結果をAccに格納する。

Accの内容と200番地の内容を加算し、結果をAccに保持

A 200
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *f*, *m*←"+"
clock5: *i*, *e*
clock6: *b*, *h*, *g*, *m*←"+"
clock7: *i*, *c*
（次のフェッチ動作へ）

Clock6, 7:
プログラムカウンタ + 1



休 憩

- ここで、少し休憩しましょう。
- また、フェッチの段階や、ロード命令、加算命令での動作を振り返って、自分の言葉で動作を説明できるようにしてみてください。
- 深呼吸したり、肩の力を抜いてから、
次のビデオに進んでください。

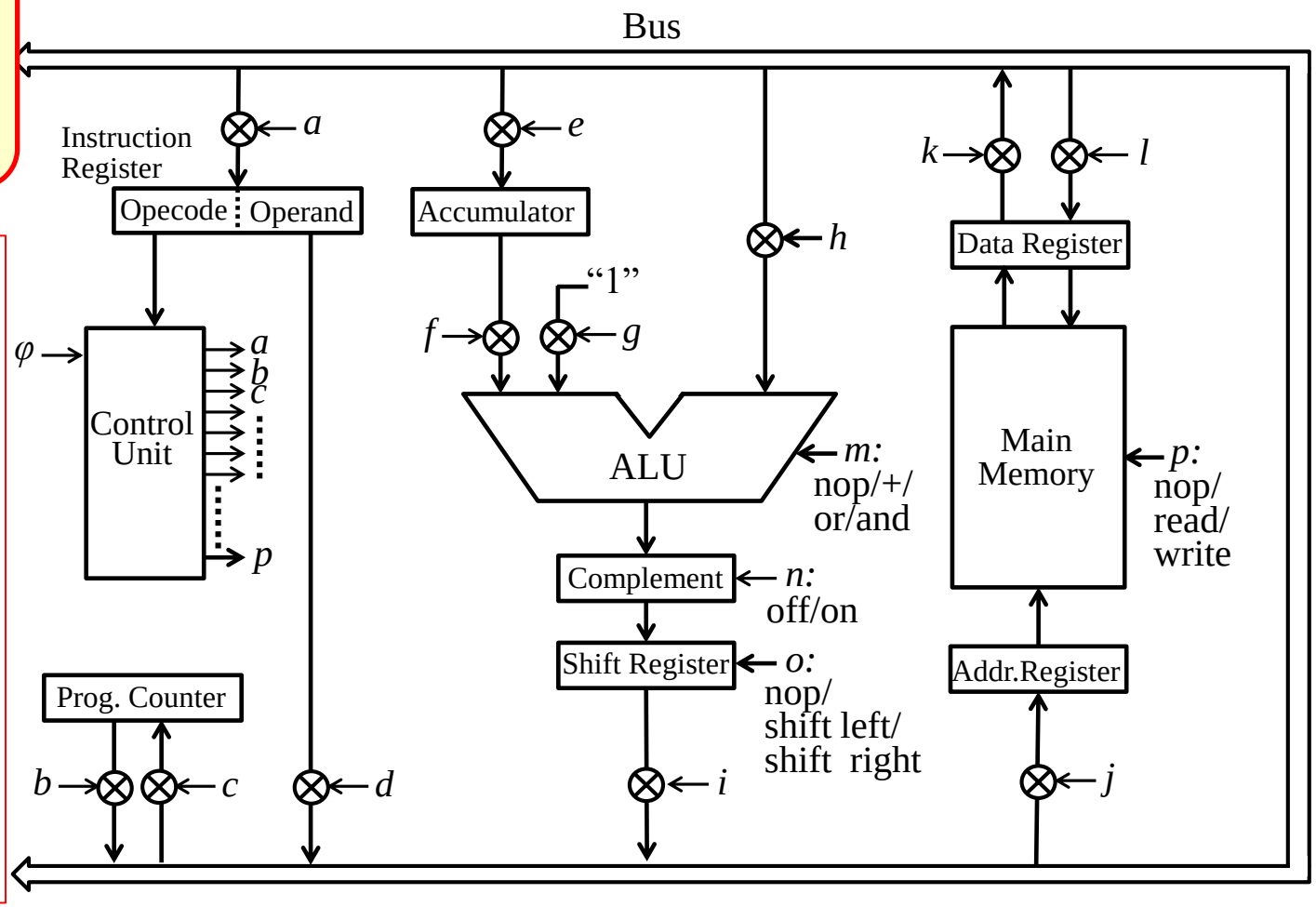
減算(Subtract)命令

・ オペランドで指定された番地のデータを読み出して
アキュムレータ(Acc)から減算し、結果をAccに格納する。

→「2の補数」を作って加算すればよい

Accの内容から300番地の内容を減算し、結果をAccに保持

SUB 300
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *n*←"on"
clock5: *i*, *h*, *f*, *m*←"+"
clock6: *i*, *h*, *g*, *m*←"+"
clock7: *i*, *e*
clock8: *b*, *h*, *g*, *m*←"+"
clock9: *i*, *c*
（次のフェッチ動作へ）



負の2進数の表現（3ビットの場合）

符号なし2進数			
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

絶対値表現			
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	0
1	0	1	-1
1	1	0	-2
1	1	1	-3

符号　絶対値

1の補数表現			
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-3
1	0	1	-2
1	1	0	-1
1	1	1	0

符号

各ビット反転→補数

2の補数表現

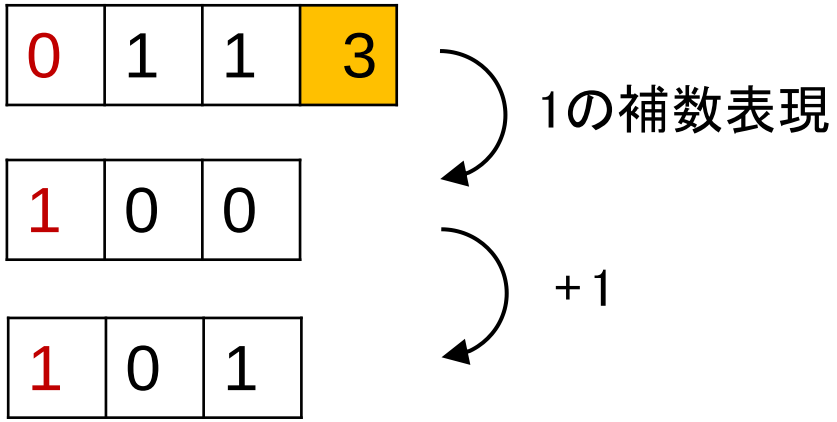
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

符号

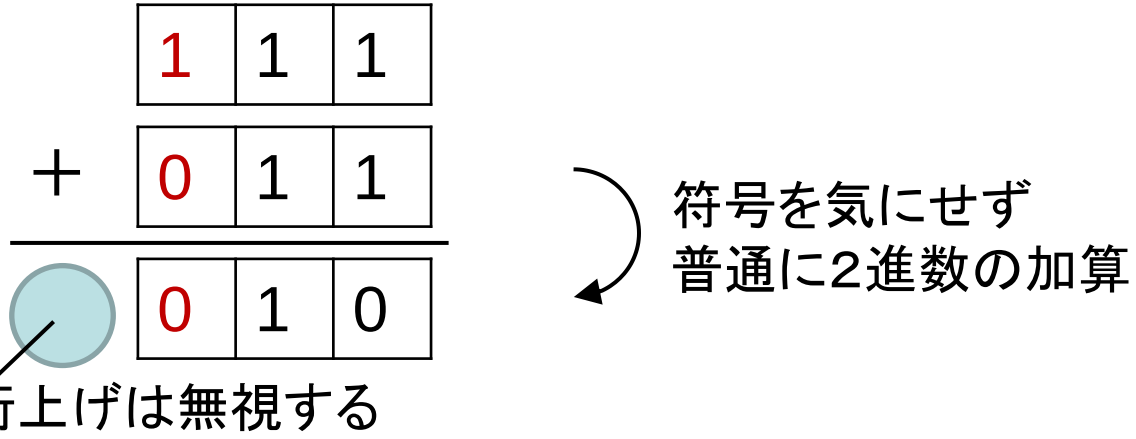
各ビットを反転 +1 →補数

負の2進数の表現（3ビットの場合）

−3（2の補数表現）は？



−1 + 3（2の補数表現）は？



2の補数表現

0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	-4
1	0	1	-3
1	1	0	-2
1	1	1	-1

符号

各ビットを反転 +1
→補数

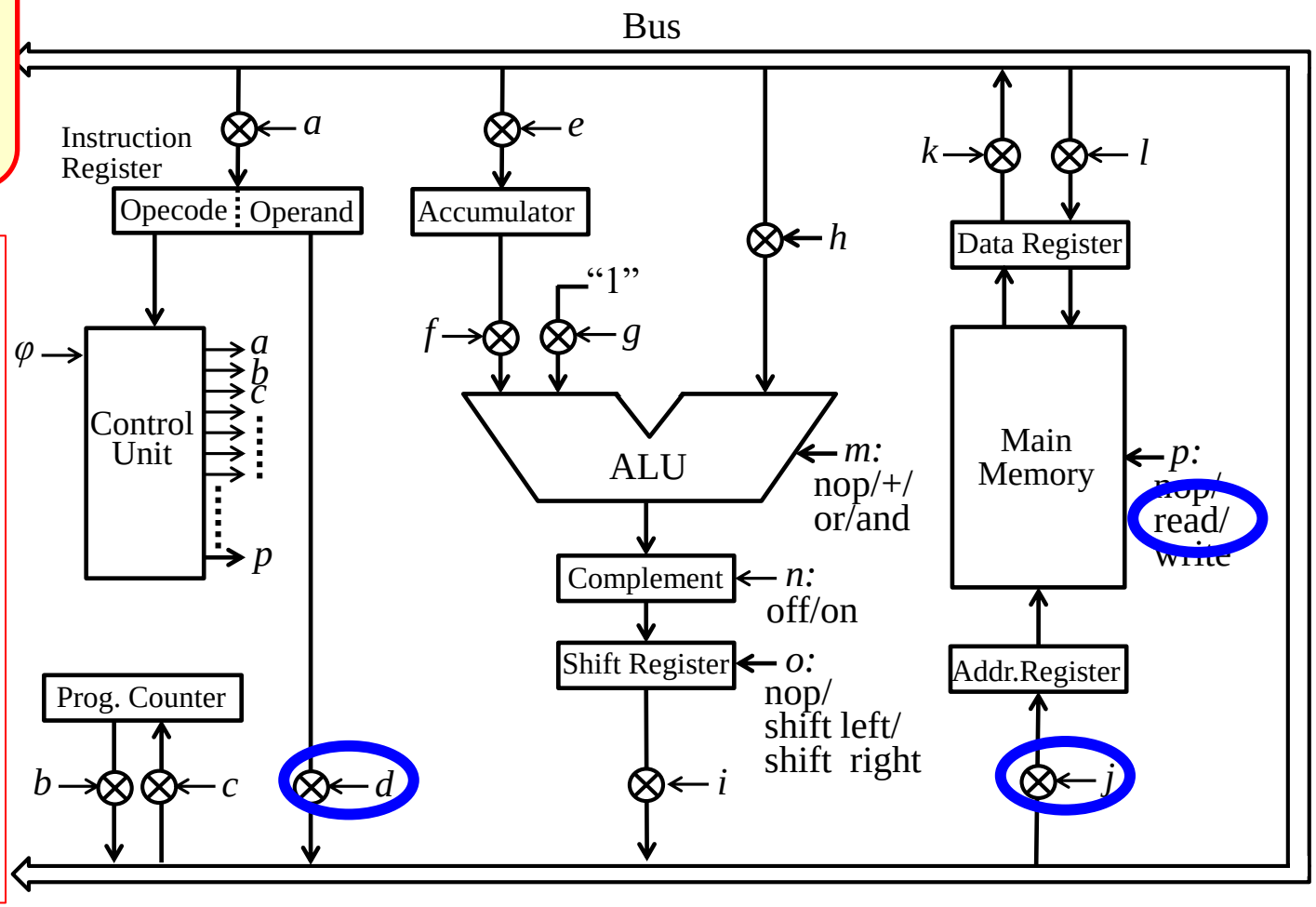
減算(Subtract)命令

・ オペランドで指定された番地のデータを読み出して
アキュムレータ(Acc)から減算し、結果をAccに格納する。

→「2の補数」を作って加算すればよい

Accの内容から300番地の内容を減算し、結果をAccに保持

SUB 300
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *n*←"on"
clock5: *i*, *h*, *f*, *m*←"+"
clock6: *i*, *h*, *g*, *m*←"+"
clock7: *i*, *e*
clock8: *b*, *h*, *g*, *m*←"+"
clock9: *i*, *c*
（次のフェッチ動作へ）



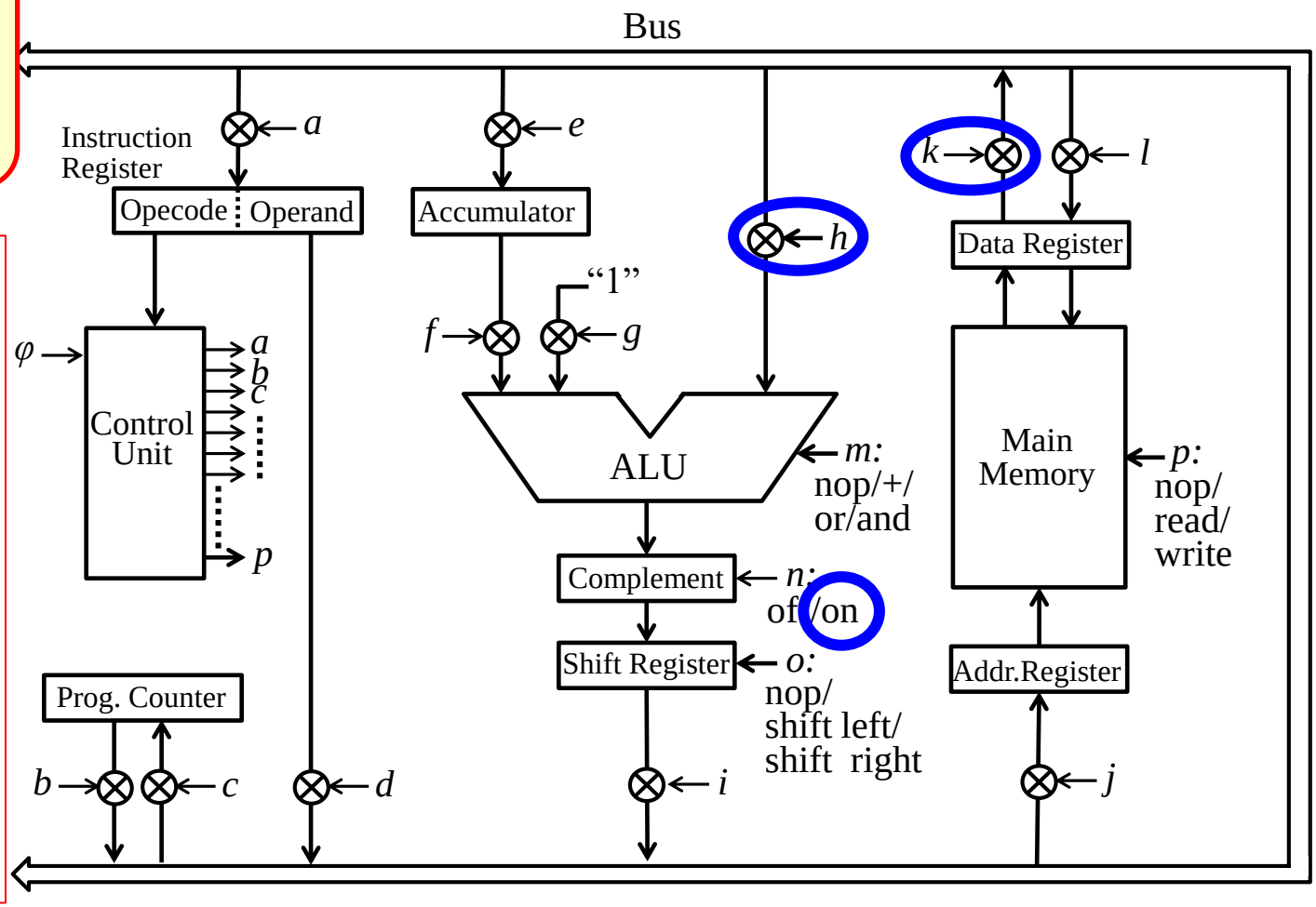
減算(Subtract)命令

・ オペランドで指定された番地のデータを読み出して
アキュムレータ(Acc)から減算し、結果をAccに格納する。

→「2の補数」を作って加算すればよい

Accの内容から300番地の内容を減算し、結果をAccに保持

SUB 300
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *n*←"on"
clock5: *i*, *h*, *f*, *m*←"+"
clock6: *i*, *h*, *g*, *m*←"+"
clock7: *i*, *e*
clock8: *b*, *h*, *g*, *m*←"+"
clock9: *i*, *c*
（次のフェッチ動作へ）



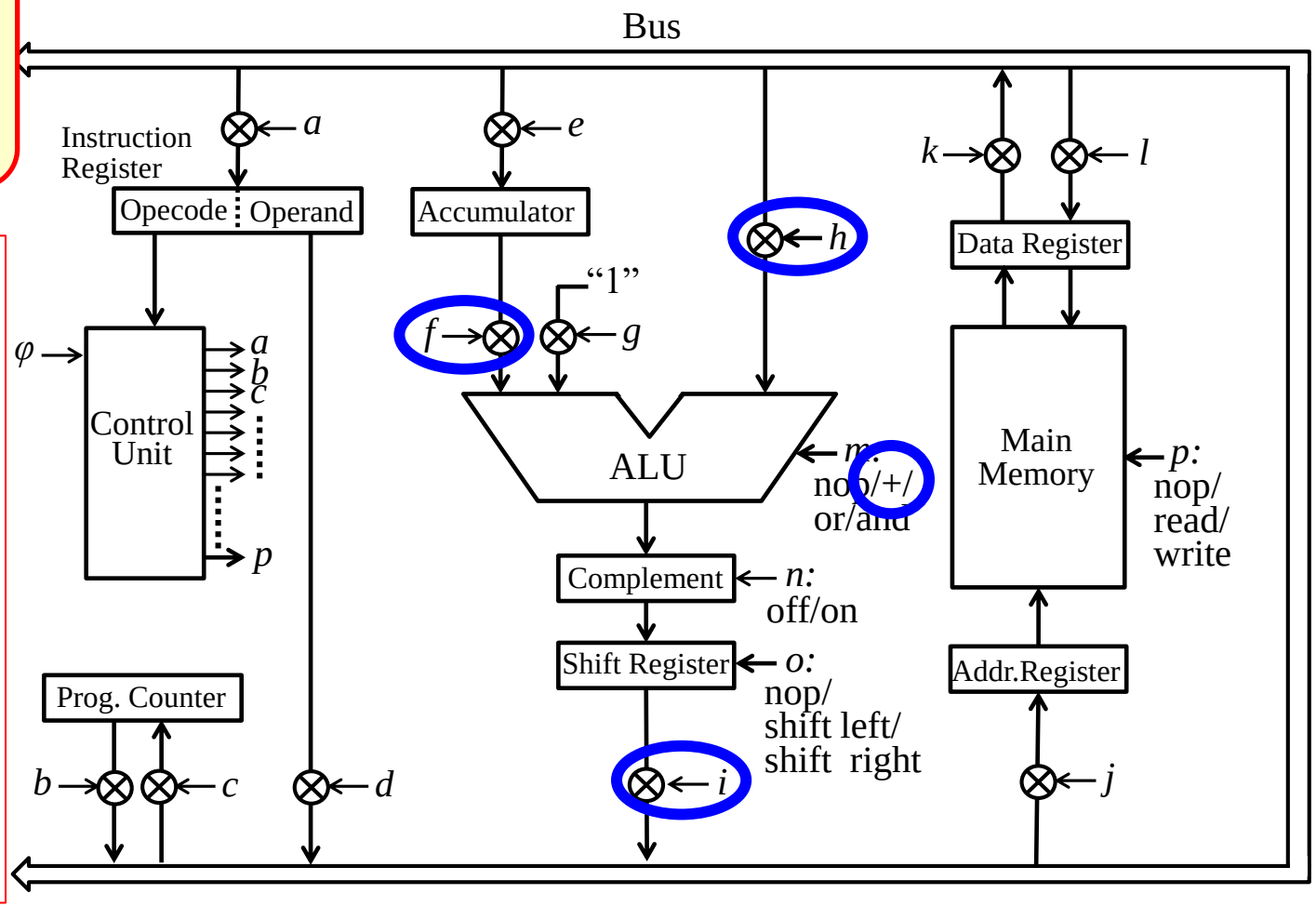
減算(Subtract)命令

・ オペランドで指定された番地のデータを読み出して
アキュムレータ(Acc)から減算し、結果をAccに格納する。

→「2の補数」を作って加算すればよい

Accの内容から300番地の内容を減算し、結果をAccに保持

SUB 300
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *n*←"on"
clock5: *i*, *h*, *f*, *m*←"+"
clock6: *i*, *h*, *g*, *m*←"+"
clock7: *i*, *e*
clock8: *b*, *h*, *g*, *m*←"+"
clock9: *i*, *c*
（次のフェッチ動作へ）



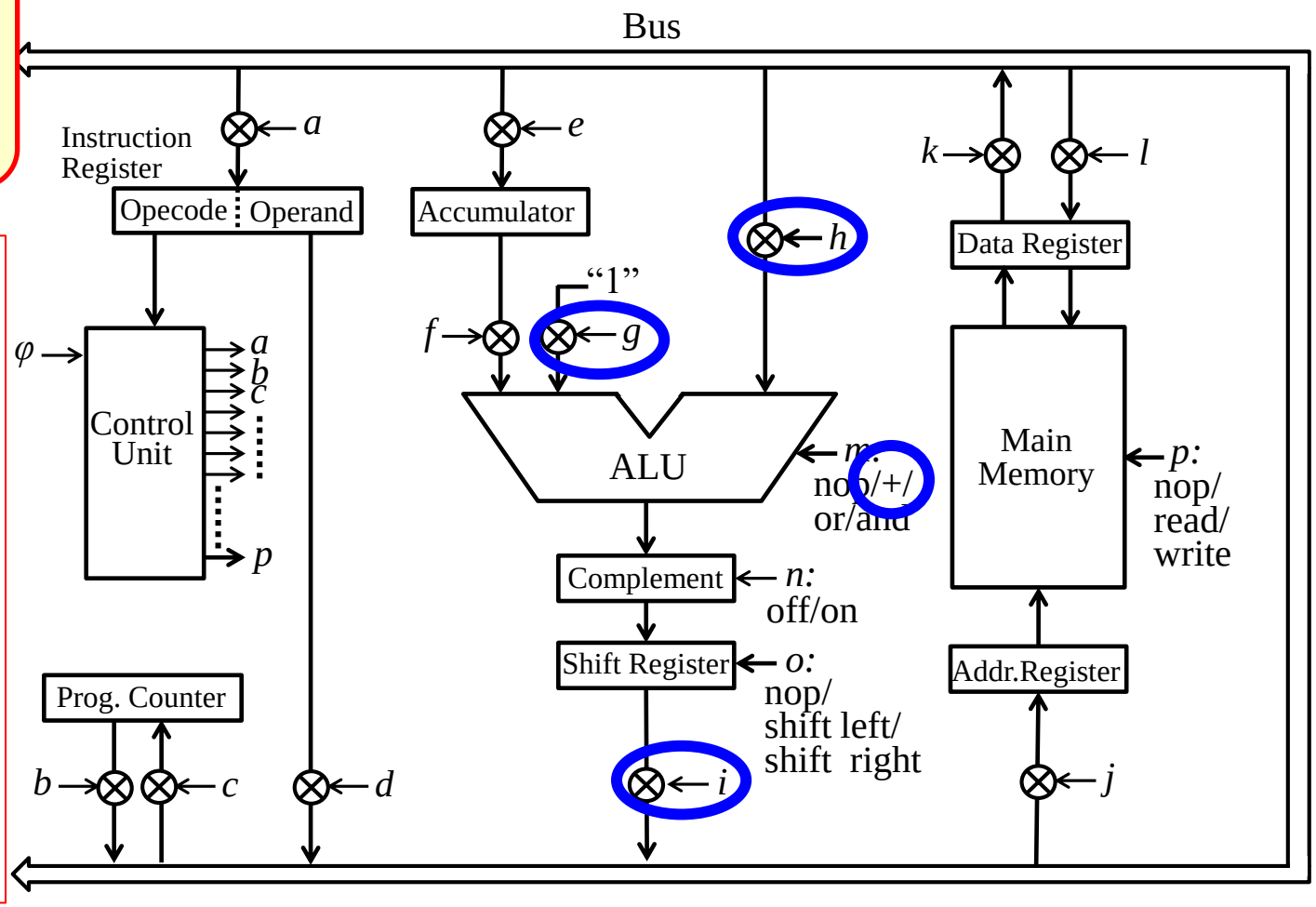
減算(Subtract)命令

・ オペランドで指定された番地のデータを読み出して
アキュムレータ(Acc)から減算し、結果をAccに格納する。

→「2の補数」を作って加算すればよい

Accの内容から300番地の内容を減算し、結果をAccに保持

SUB 300
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *n*←"on"
clock5: *i*, *h*, *f*, *m*←"+"
clock6: *i*, *h*, *g*, *m*←"+"
clock7: *i*, *e*
clock8: *b*, *h*, *g*, *m*←"+"
clock9: *i*, *c*
（次のフェッチ動作へ）



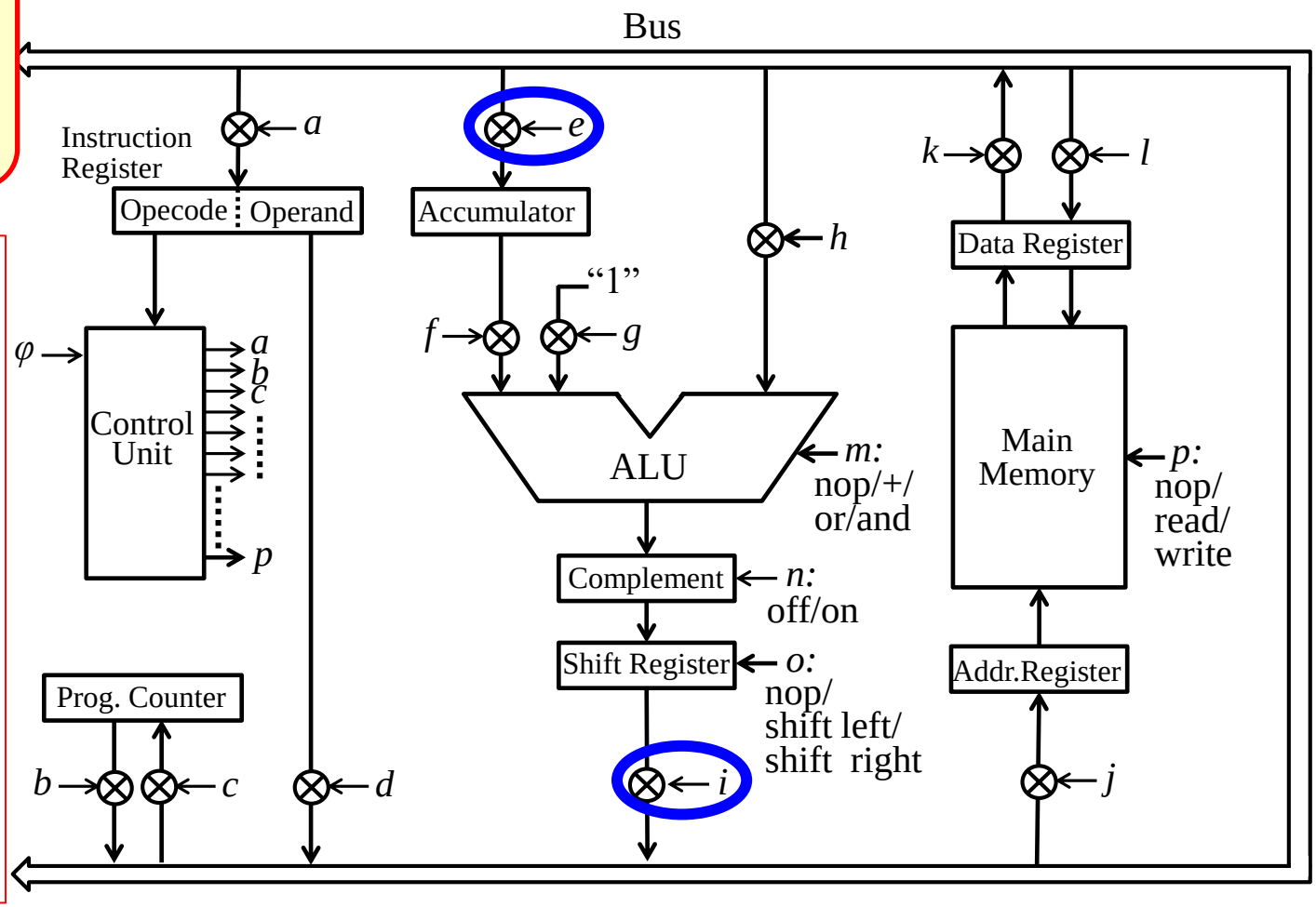
減算(Subtract)命令

・ オペランドで指定された番地のデータを読み出して
アキュムレータ(Acc)から減算し、結果をAccに格納する。

→「2の補数」を作って加算すればよい

Accの内容から300番地の内容を減算し、結果をAccに保持

SUB 300
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *n*←"on"
clock5: *i*, *h*, *f*, *m*←"+"
clock6: *i*, *h*, *g*, *m*←"+"
clock7: *i*, *e*
clock8: *b*, *h*, *g*, *m*←"+"
clock9: *i*, *c*
（次のフェッチ動作へ）



減算(Subtract)命令

・ オペランドで指定された番地のデータを読み出して
アキュムレータ(Acc)から減算し、結果をAccに格納する。

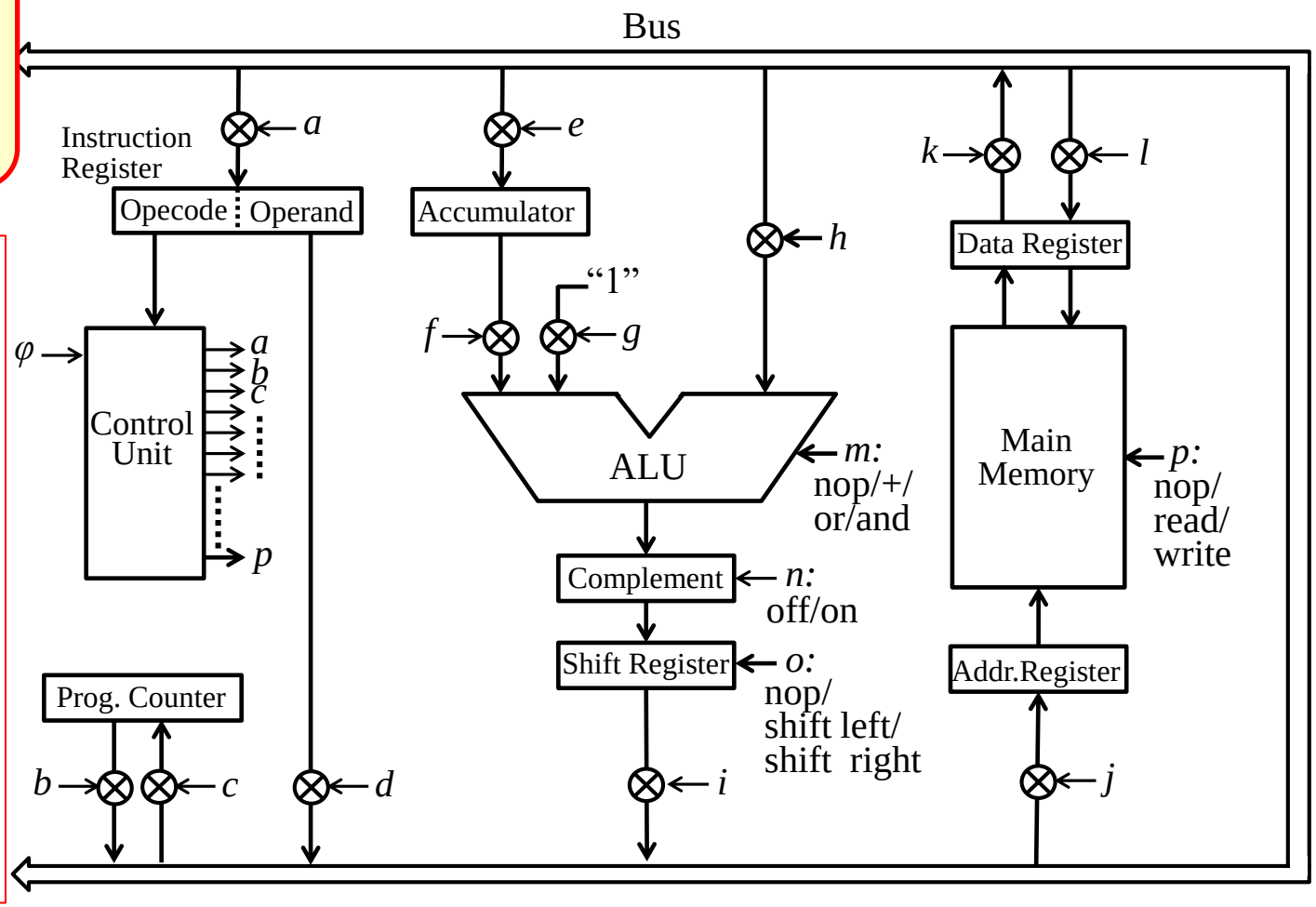
→「2の補数」を作って加算すればよい

Accの内容から300番地の内容を減算し、結果をAccに保持

SUB 300
clock3: *d*, *j*, *p*←"read"
clock4: *k*, *h*, *n*←"on"
clock5: *i*, *h*, *f*, *m*←"+"

Clock8, 9:
プログラムカウンタ + 1

clock8: *b*, *h*, *g*, *m*←"+"
clock9: *i*, *c*
（次のフェッチ動作へ）

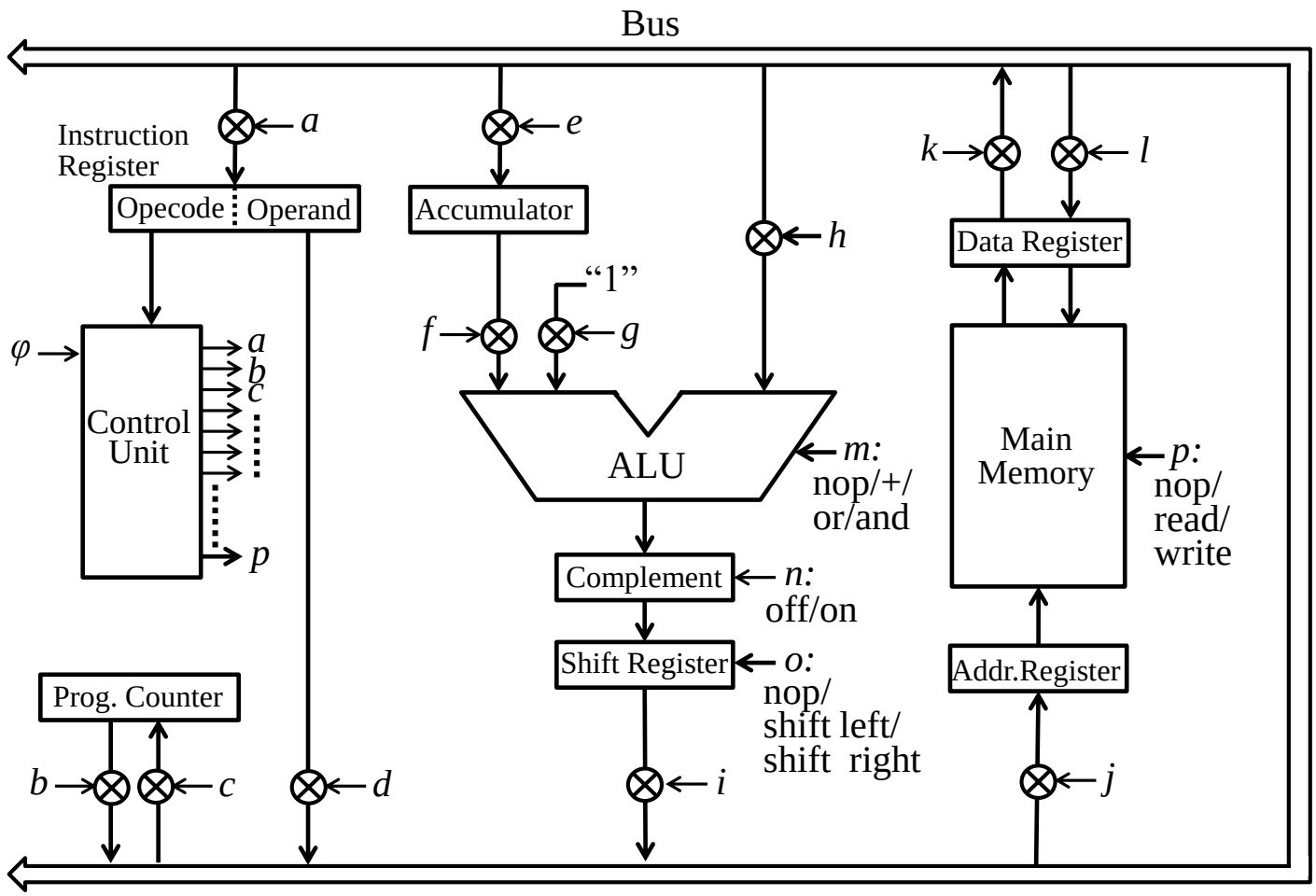


分岐(Jump)命令

- オペランドで指定された番地に実行を移す。
  → 次の機械語命令は指定番地から読み出す

500番地に
実行を移す

JMP 500
clock3: *d*, *c*
（次のフェッチ動作へ）

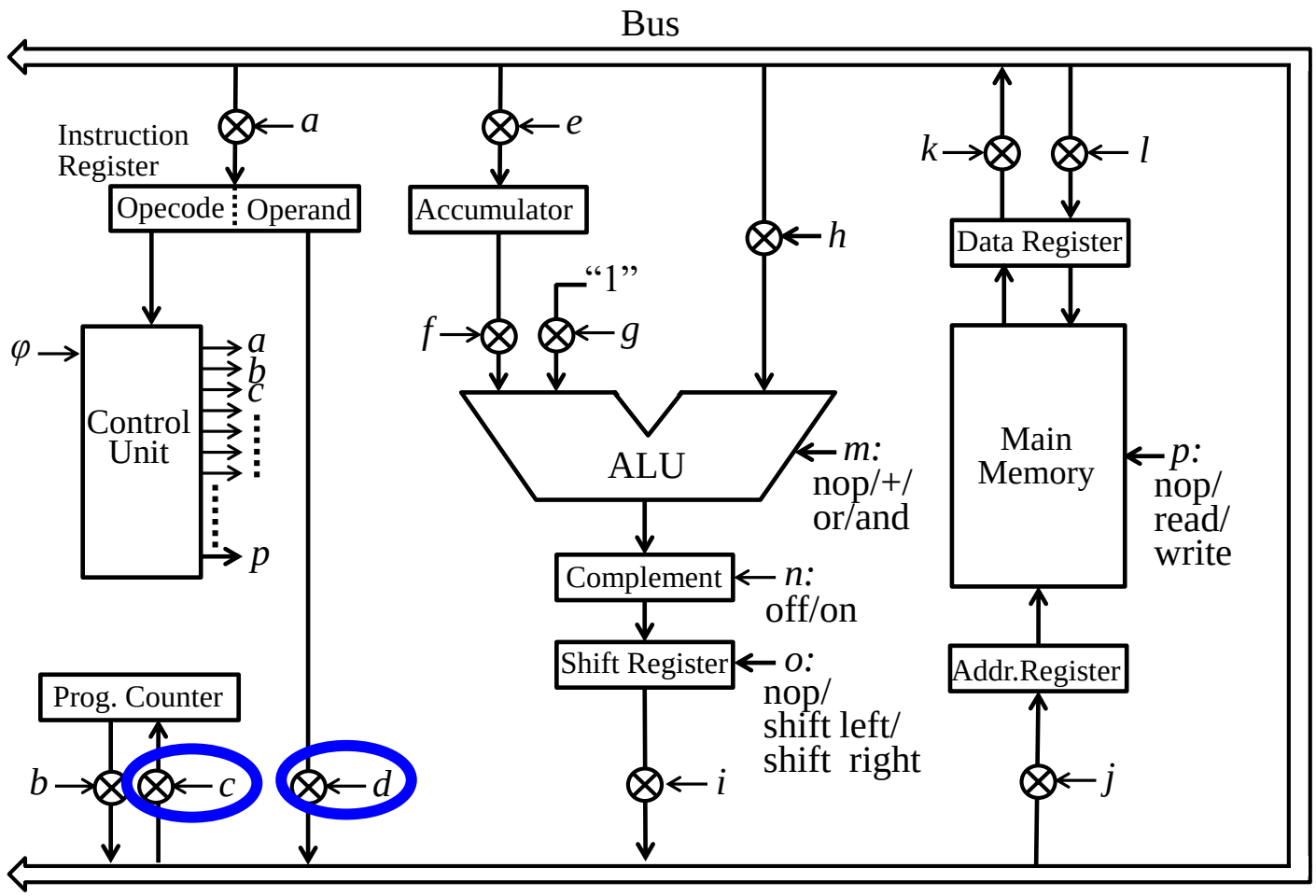


分岐(Jump)命令

- オペランドで指定された番地に実行を移す。
  → 次の機械語命令は指定番地から読み出す

500番地に
実行を移す

JMP 500
clock3: *d*, *c*
（次のフェッチ動作へ）

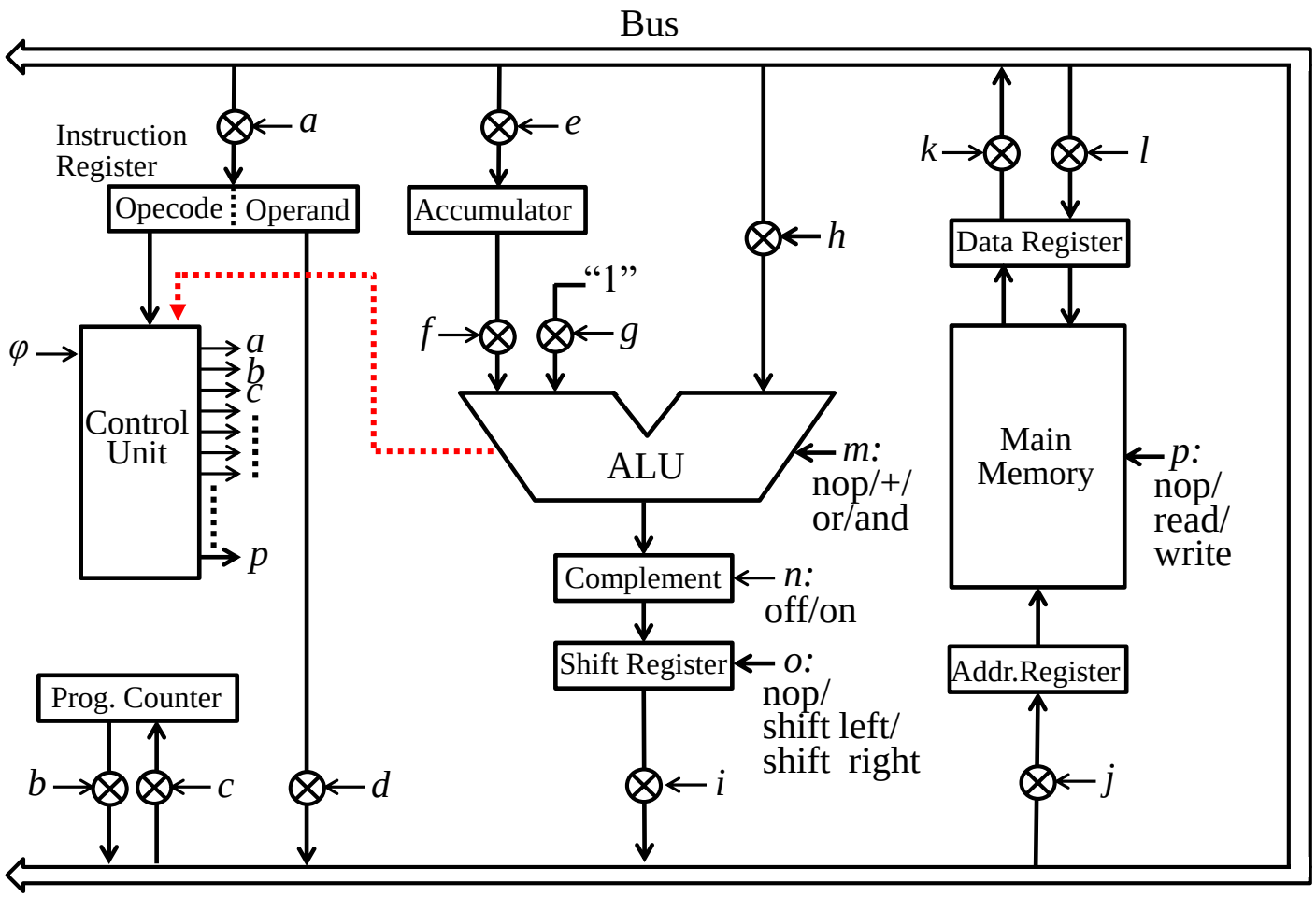


条件分岐(Conditional Jump)命令

- 直前の演算結果に依存して分岐命令を実行するか決める
  - 場合分けや繰り返しなどの制御が可能になる

直前の命令の演算結果がゼロでなければ550番地へ飛ぶ

JMP NZ 550
clock3: b, h, g, m←“+”
clock4: (Z? i : d), c
（次のフェッチ動作へ）

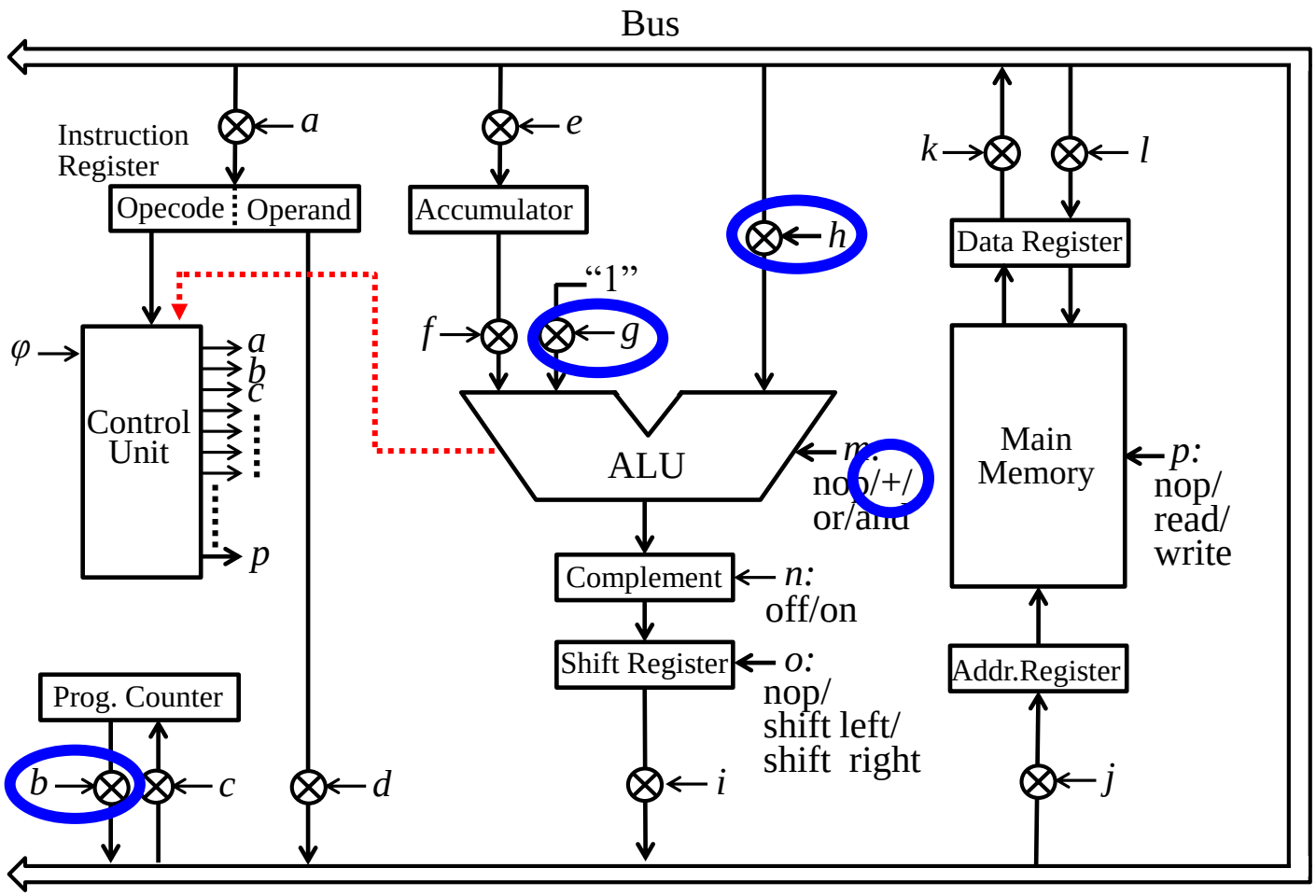


条件分岐(Conditional Jump)命令

- 直前の演算結果に依存して分岐命令を実行するか決める
  - 場合分けや繰り返しなどの制御が可能になる

直前の命令の演算結果がゼロでなければ550番地へ飛ぶ

JMP NZ 550
clock3: $b, h, g, m \leftarrow$ "+"
clock4: (Z? i : d), c
（次のフェッチ動作へ）

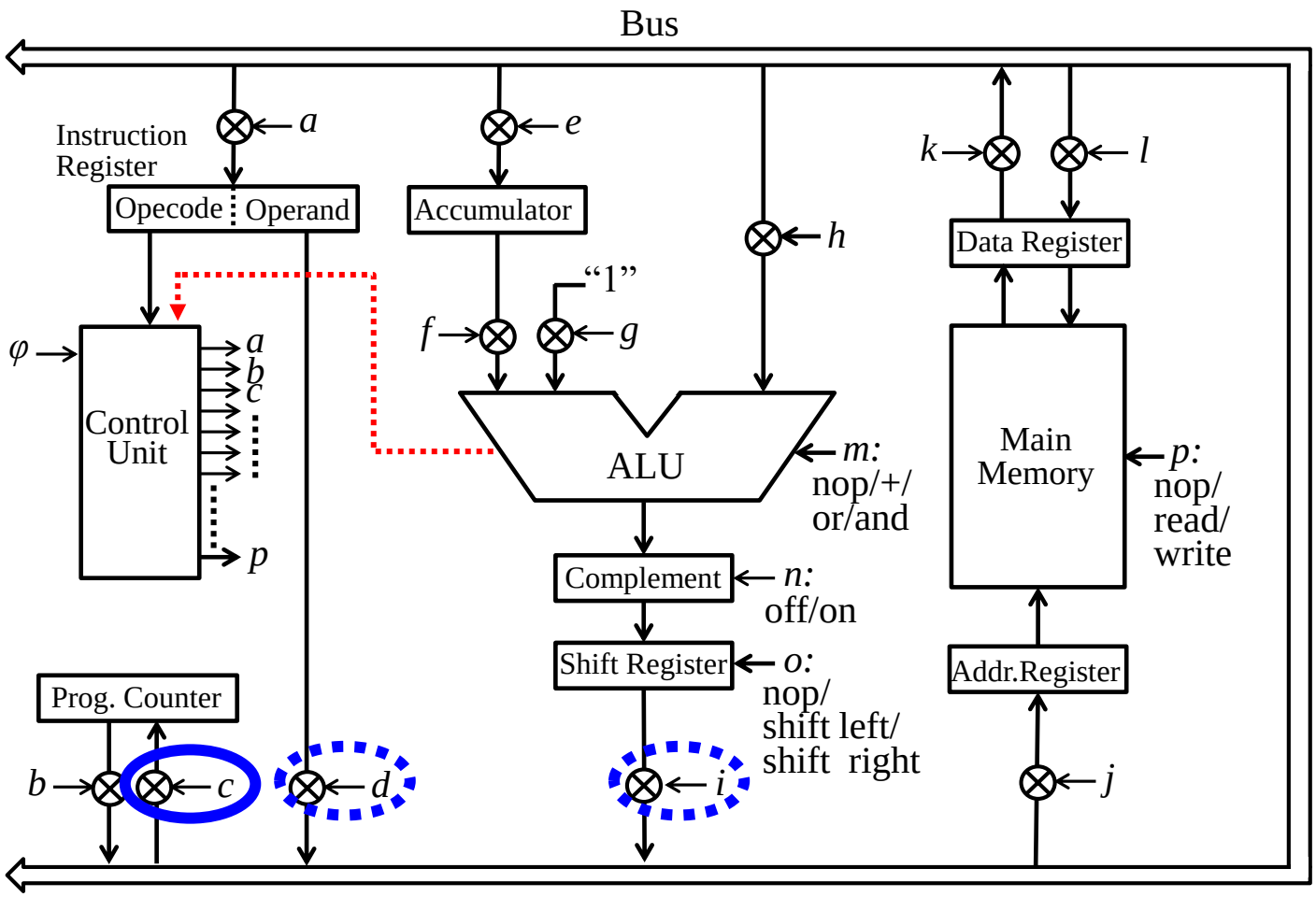


条件分岐(Conditional Jump)命令

- 直前の演算結果に依存して分岐命令を実行するか決める
  - 場合分けや繰り返しなどの制御が可能になる

直前の命令の演算結果がゼロでなければ550番地へ飛ぶ

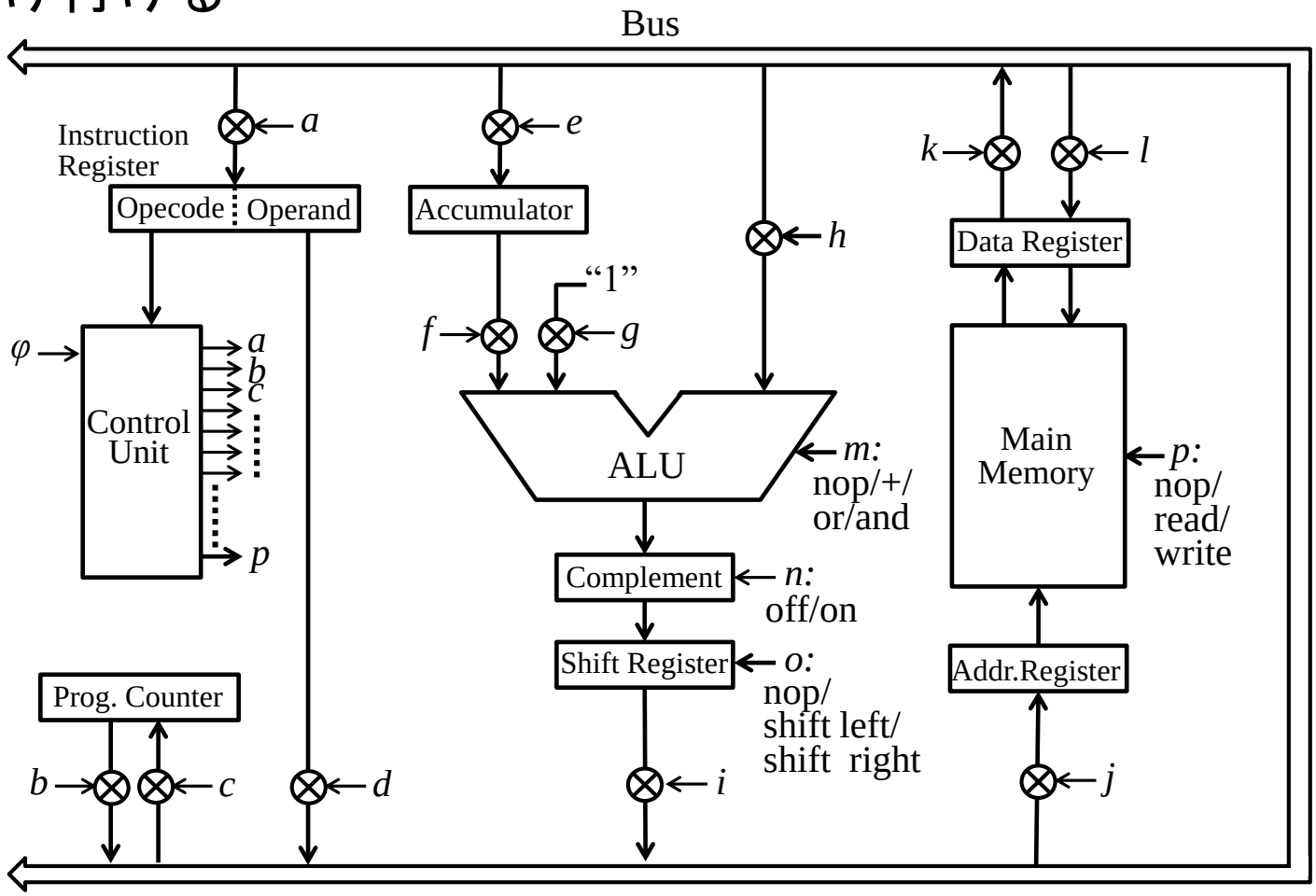
JMP NZ 550
clock3: $b, h, g, m \leftarrow$ "+"
clock4: (Z? i : d), c
（次のフェッチ動作へ）



停止(Halt)命令

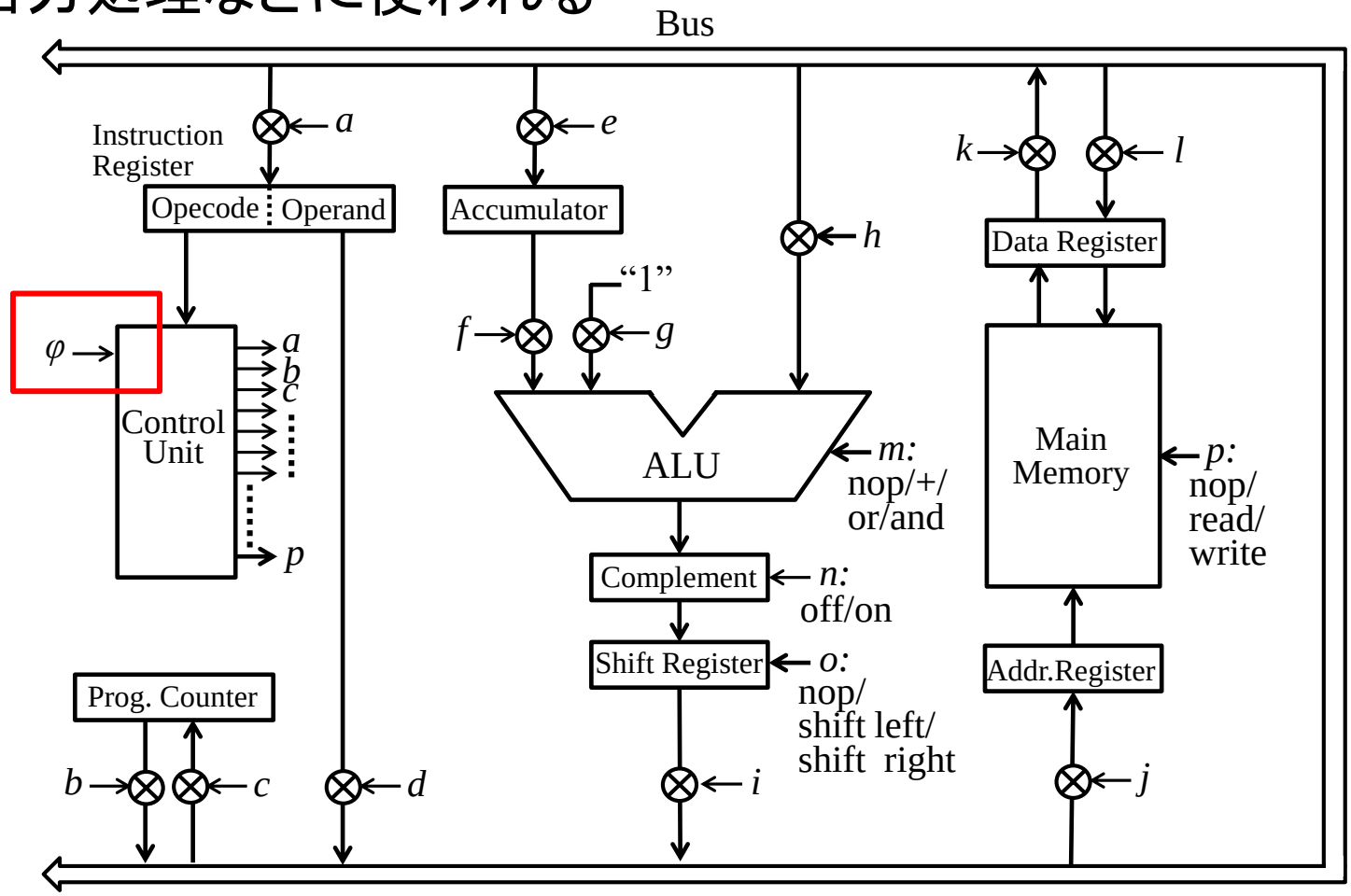
- プログラムの実行を停止する(電源オフではない)
  - 何もせずフェッチ動作を繰り返す無限ループ状態に入る
  - 「割り込み」は受け付ける

HALT
(次のフェッチ動作へ)



割り込み(Interrupt)動作

- 外部からの信号で強制的に実行番地を移す機能
  - 分岐先はあらかじめ指定されている
  - タイマーや、入出力処理などに使われる



今回のまとめ

機械語命令と内部動作

- 基本的・典型的なアーキテクチャの構成
- 一般的な機械語命令の形式
- 機械語命令に対するレジスタ転送レベルの内部動作
  - フェッチ動作
  - ロード命令
  - 加算命令
  - 減算命令(→ 補数表現)
  - ストア命令
  - 条件分岐命令
  - 停止命令と割り込み動作