Архитектура вычислительных систем

Лекция 5.

Яревский Е.А.

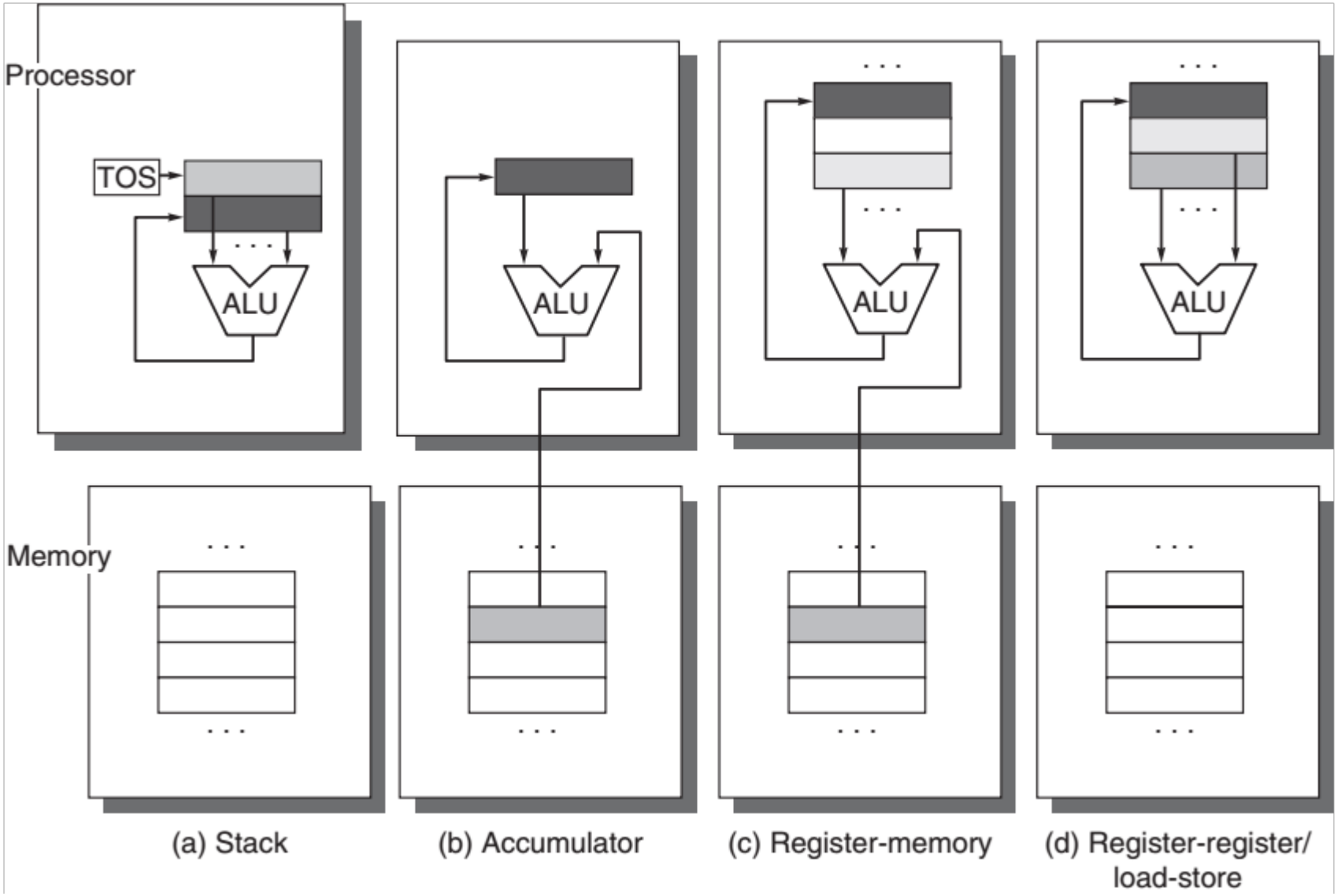
Кафедра вычислительной физики

Уровень архитектуры набора команд (Instruction Set Architecture)

Характеристики уровня:

1) Класс архитектуры
(регистр-память или load-store (чтение-запись), число операндов и их размещение)
2) Адресация памяти
(наименьшая адресуемая ячейка, допустимые объекты, требование выравнивания)
3) Типы адресации
(абсолютная, регистровая, непосредственная, смещение, индексирование и т.д.)
4) Типы и размеры операндов
(8 бит, 16 бит, и т.д.)
5) Операции
(передачи данных, арифметико-логические, с плавающей запятой, управляющие потоком команд)
6) Организация управляющих операций
(типы переходов + размещение тестируемых элементов + размещение адреса возврата)
7) Формат кодируемой инструкции
(поля, фиксированная/переменная длина инструкций)

Классификация наборов инструкций



Stack	Accumulator	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

Примеры кодов для действия C=A+B для процессоров различной организации.

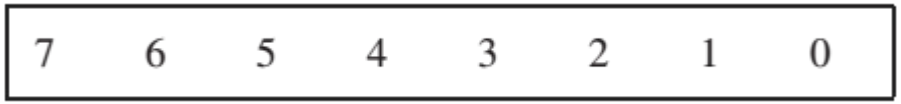
Процессоры архитектур регистров общего назначения (General Purpose Registers, GPR) используют только явные операнды команд, которые могут находиться в регистрах или в памяти.
Преимущество регистров в скорости и эффективности (с точки зрения компиляторов) привело к доминированию процессоров GPR.

Две важные характеристики делят GPR процессоры на группы: число операндов типичных арифметических и логических операций и число операндов, размещенных в памяти. Есть типы с трехоперандными командами – два операнда-источника и операнд-приемник. В командах двухоперандного типа один из операндов-источников является и операндом-приемником. По числу операндов, размещенных в памяти, возможны колебания от нуля до трех.

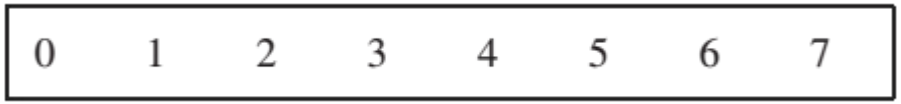
Адресация памяти

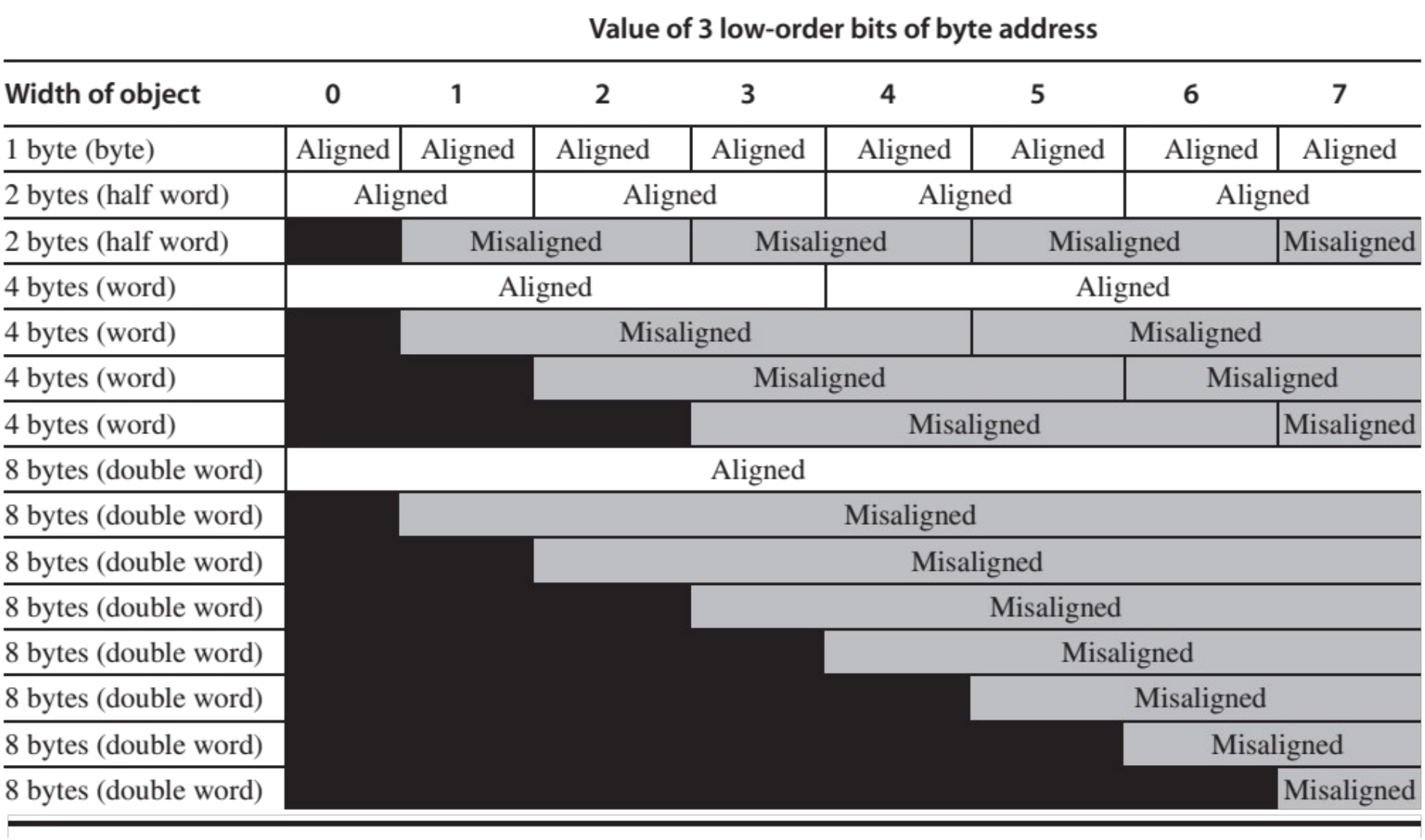
Архитектура набора команд должна определять правила, по которым объекты извлекаются из оперативной памяти. Как правило, используется байтовая адресация памяти. Кроме байта, используются полуслова длиной 16 бит, слова 32 бита и, часто, двойные слова длиной 64 бита. (Терминология x86 отличается от этой!)

Есть два различных соглашения по размещению байтов внутри слова: «Little Endian» и «Big Endian».
При порядке «Little Endian» адрес 8-ми байтного двойного слова совпадает с адресом наименее значимого байта:



В порядке «Big Endian» адрес двойного слова совпадает с адресом наиболее значимого байта:





Выравнивание объектов, больших байта.

При извлечении из адреса A объекта длиной s байтов должно выполняться условие **A mod s = 0**. Это ограничение позволяет избежать усложнения аппаратной составляющей и накладок при непреднамеренном одновременном обращении к одной ячейке с различной длиной элементов. Даже на компьютерах, не требующих выравнивания, «выровненные» программы работают быстрее.

Типы адресации

Тип адресации	Пример команды	Значение	Когда используется
Регистровая	Add R4, R3	Reg[4]<-Reg[4]+Reg[3]	Когда значения в регистрах
Непосредственная	Add R4, #3	Reg[4]<-Reg[4]+3	Для констант
Смещение	Add R4, 100(R1)	Reg[4]<-Reg[4]+ Mem[100+R[1]]	Для доступа к локальным переменным,
Регистровая косвенная	Add R4, (R1)	Reg[4]<-Reg[4] +Mem[Reg[1]]	По указателю или по вычисленному адресу
Индексированная	Add R3,(R1+R2)	Reg[3]<-Reg[3] +Mem[Reg[1]+Reg[2]	При обращении к массивам: R1 – начало массива, R2 – индекс элемента массива
Абсолютная	Add R1, (1001)	Reg[1]<-Reg[1] +Mem[1001]	При обращении к статическим данным; константа адреса должна быть большой
Косвенная из памяти	Add R1, @(R3)	Reg[1]<-Reg[1] +Mem[Mem[Reg[3]]]	При кратном разыменовании
Автоинкрементная	Add R1, (R2)+	Reg[1]<-Reg[1] +Mem[Reg[2] Reg[2]<-Reg[2]+d	При проходе по массиву в цикле; d – длина элемента массива
Автодекрементная	Add R1, -(R2)	Reg[2]<-Reg[2]-d Reg[1]<-Reg[1] +Mem[Reg[2]	Также, как автоинкремент. И для имплементации стека.
Масштабированная	Add R1,100(R2) [R3]	Reg[1]<-Reg[1] +Mem[100 + Reg[2] +Reg[3] * d]	Для индексирования массивов

Типы и размеры операндов

Наиболее распространенные типы:
integer (целый), **single-precision floating point** (вещественный одинарной точности) **character** (символьный) и т.д. – типы, эффективно определяемые размером.
Общие типы операндов включают **символьный** (8 бит), полуслово (16 бит), слово (32 бита), вещественный одинарной точности (одно слово) и вещественный двойной точности (2 слова).
Целые обычно представляются двоичным кодом с дополнением до двух.
Символьный тип использует ASCII и 16-битный Юникод.
Представление вещественных типов следует стандартам IEEE 754.

Некоторые архитектуры поддерживают операции над символьными строками, трактуя их как последовательность символьных байтов.
Типичными операциями являются копирование и сравнение.

Для бизнес-приложений иногда поддерживается десятичный формат в виде упакованных десятичных или двоично-десятичных: 4 бита используются для кодирования значений от 0 до 9 и две десятичные цифры упаковываются в 1 байт. (неточностью двоичных вычислений, точных в десятичной системе счисления).

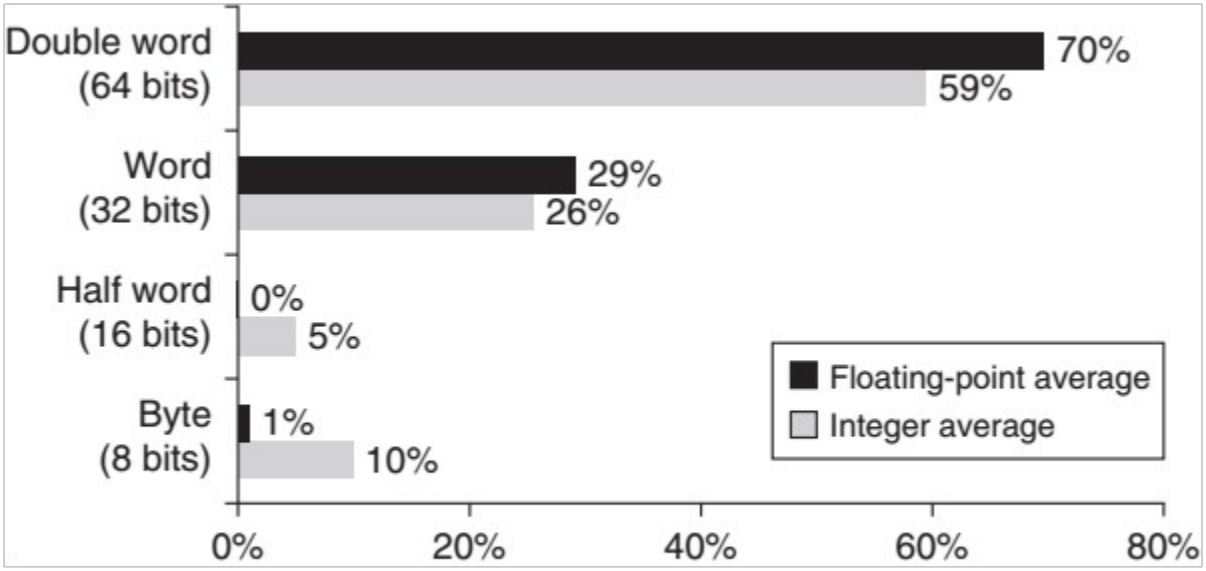


Figure A.11 Distribution of data accesses by size for the benchmark programs.

Операции набора инструкций

Тип операции	Примеры
Арифметический и логический	Целочисленные арифметические и логические операции: Сложение и вычитание, умножение и деление, логические И и ИЛИ
Передачи данных	Load-store (загрузка-сохранение) и копирование (move) в системах с адресуемой памятью)
Управляющий	Ветвления, переходы, вызов процедур и возврат, прерывания
Системный	Управление виртуальной памятью, запрос на системные вызовы
С плавающей точкой	Арифметические действия с плавающей точкой
Десятичный	Десятичная арифметика, преобразования в десятичные строки
Строковый	Копирование и сравнение строк, поиск
Графический	Пиксельные и векторные операции, сжатие/восстановление

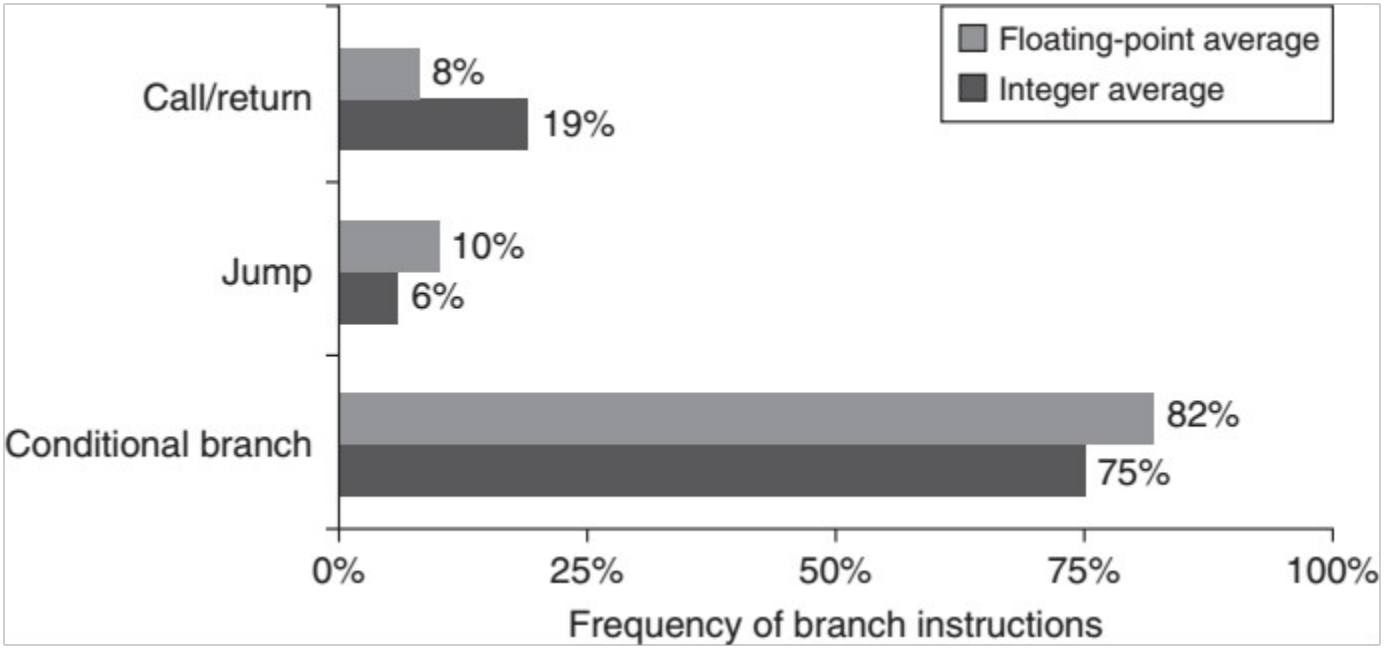
Операции набора инструкций

Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
Total		**96%**

Figure A.13 The top 10 instructions for the 80x86. Simple instructions dominate this list and are responsible for 96% of the instructions executed. These percentages are the average of the five SPECint92 programs.

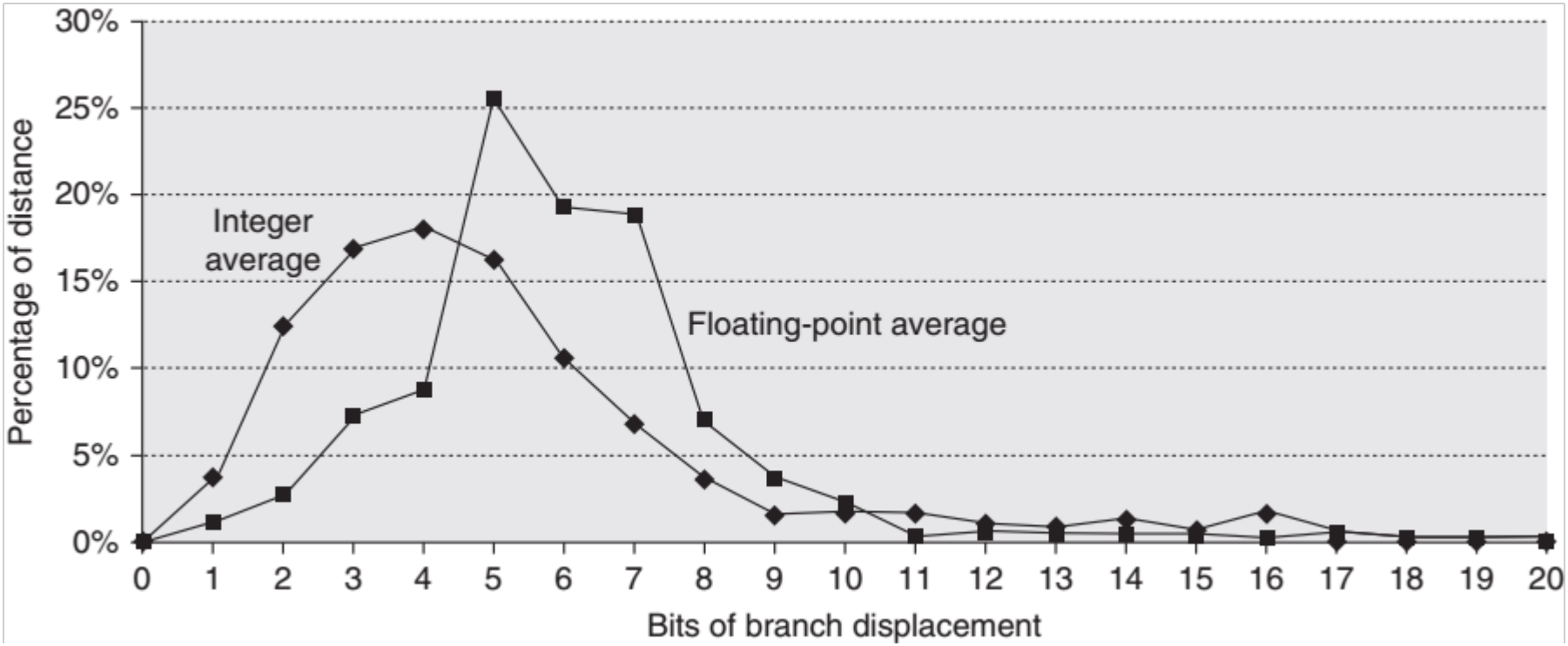
Организация управляющих операций

– переходы (безусловные)
– ветвления (условные)
– вызов/возврат из процедур.



Нужно задавать адрес назначения. Очень часто – смещение относительно указателя команд (статический режим).
Динамически – через значение в регистре, или с другой модой адресации.

Организация управляющих операций

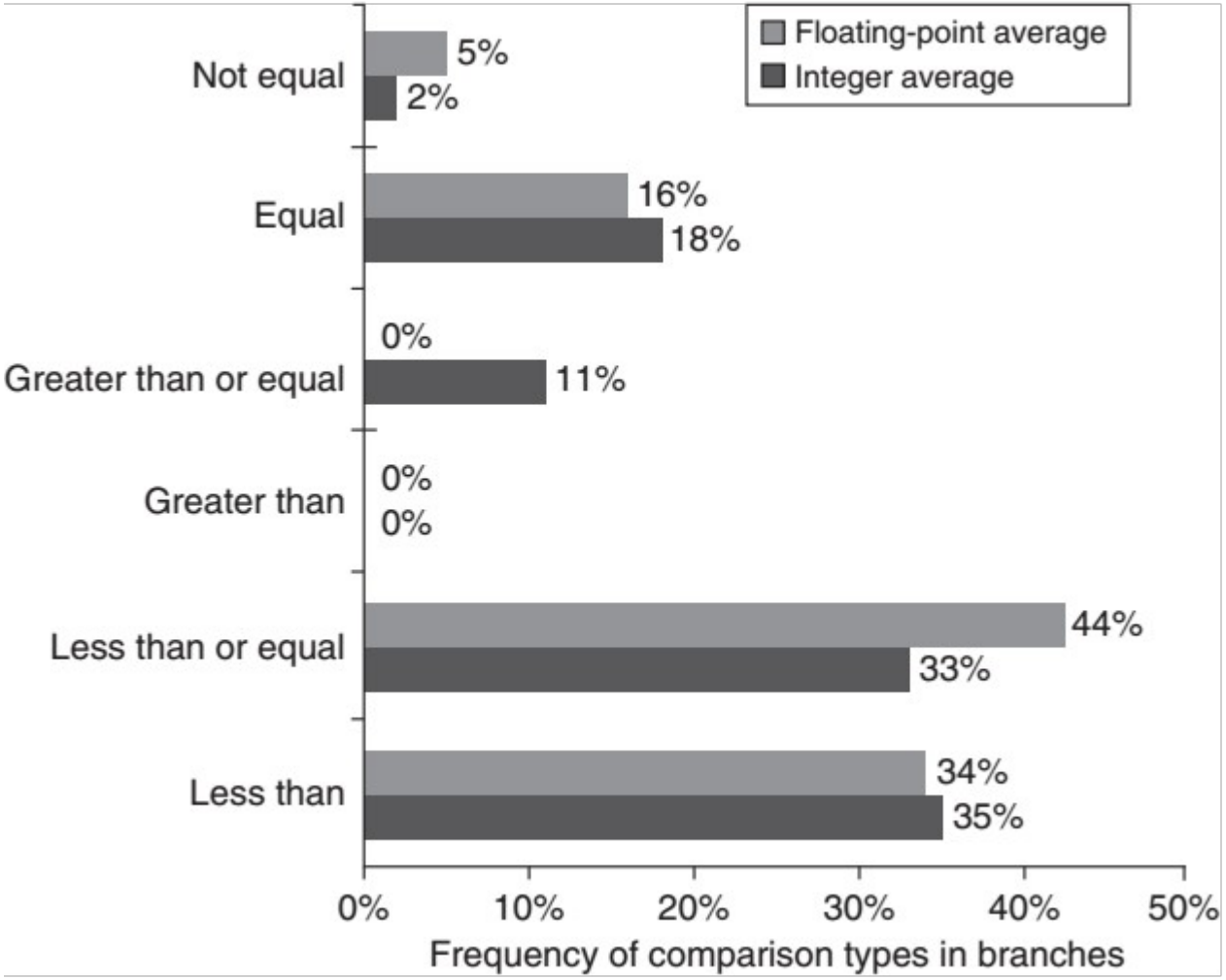


Выбор оптимального количества бит на кодирование смещения позволяет оптимизировать длину команд.

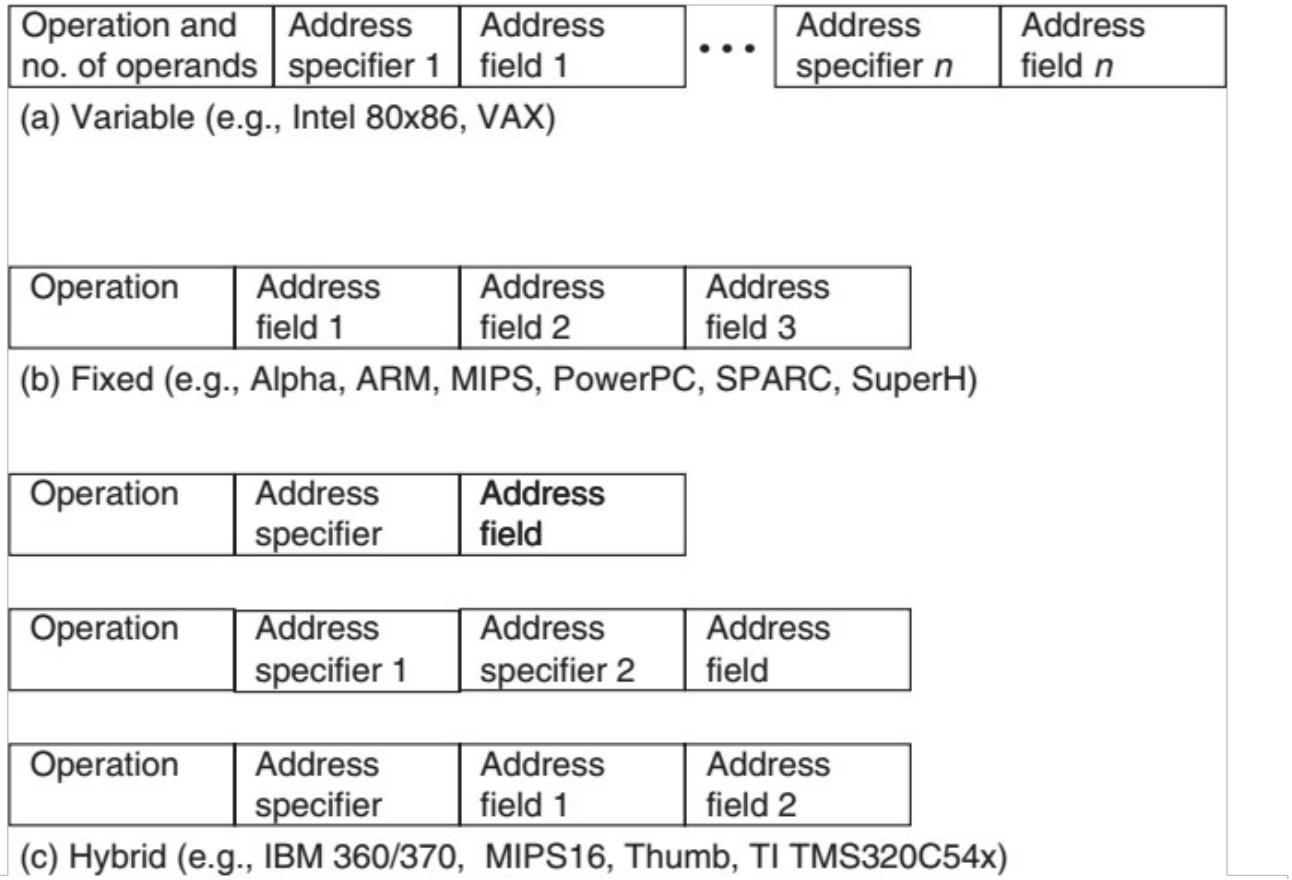
Варианты организации ветвления

Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Частота различных типов условий ветвления



Формат кодируемой инструкции



Три популярных формата инструкции: вариабельный, фиксированный и гибридный.

Определяющее значение имеет соотношение диапазона типов адресации со степенью независимости кода операции от типа адресации. Если операндов и типов адресации много, то такой набор комбинаций требует отдельного спецификатора адреса для каждого операнда. Противоположный вариант: load-store компьютеры только с одним операндом в памяти и одним-двумя типами адресации. В этом случае тип адресации должен быть включен в код операции.