Jonas B.

Systems Programming and Computer Architecture

C

- Static: value bleibt unverändert zwischen Calls
- Char: 1 byte
- Short: 2 bytes
- int: 4 bytes
- long: 8 bytes
- float: 4 bytes
- double: 8 bytes

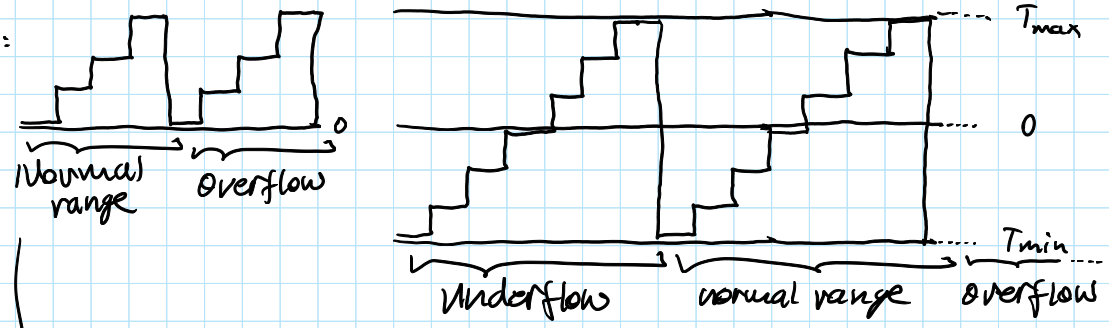
Integers — Floating point numbers

- Boolean: · 0: false
  · !0: true (anything other than 0)
- Casting: the bits don't change, only the interpretation
- Arrays: C-compiler does not check array bounds
- Strings: · C has no real string-type
  · Array of chars, terminated with null "\0"
  · char str[6] = {"h", "e", "l", "l", "o", "\0"};
  · char str[6] = "hello";
- Bit-level-operator: · And & / · Or | / · XOR: ^ / · NOT ~
- Logic-operator: · And && / · Or || / · Not!
- Arithmetic shift: If it's a signed integer, MSB (→ sign bit) is replicated back
- Logical shift: If it's an unsigned integer, 0 are filled

Numbers

- Two's complement: · Most significant bit ≙ sign bit: 1=negativ / 0=positive
  · größte Zahl = 011...1 ≙ T_{max}
  · Zero = 00...0
  · kleinste Zahl = 100...0 ≙ T_{min}
  · Reverse: 1. invert all bits
    2. Add 1 to the right (=lsb)

- Unsigned numbers: · größte Zahl: 11...1 } U_{max}
  · kleinste Zahl: 00...0 } U_{min}
- Mixing signed/unsigned: · signed → unsigned
  · unsigned → unsigned
  ↳ -1 > 0$_{unsigned}$ => true (-1_{signed} → 100...0 → Big unsigned)
- Unsigned divide: $x / 2^k = x >> k$
- Signed divide: · $x / 2^k = x >> k$ for $x > 0$ } Logical shift
  · $x / 2^k = x + (2^k - 1) >> k$ for $x < 0$ } Arithmetic shift
- Overflow:

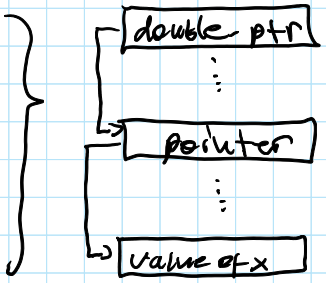


↳ We forget the carry bit and treat the remaining bits as a normal number
↳ Causes over/underflow

Pointers

- Pointers correspond to memory-addresses
- Functions have pointers as well
- &x: address >< *x: value at address
- type *name; (points somewhere in the memory space)
  ↳ Illegal to dereference it, because it points to NULL (0x00...0)
- type *name //declares a pointer
- int test = 5
- name = &test
- *test = 10

- Array: Arrays are treated as a pointer to the first element (A[i] ≙ *(A+i))
- Size: sizeof(type) or sizeof(value)
- Double pointer: · Pointer to a pointer
  · int ** double_pointer
  · int* pointer
  · pointer = &x
  · double_pointer = pointer



- int-pointer zeigt auf word line (Byte-wise access)
- char-pointer zeigt auf erstes Bit (Bit-wise access)

Passing values

- Passing by reference: Normally only objects
- Passing by value: primitives like int
- In C, we can use pointers to get a psendo pass by ref.
  ↳ Callee still receives a copy, but now it's a pointer
- A pointer can point to NULL, a reference not

Memory

- Global variables: · allocated when program is run
  · deallocated when program exits
- Local variables: · allocated when method is called
  · deallocated when method exits
- C has no garbage collector: explicetly allocate/deallocate memory
- Malloc: · void* malloc(size)
  · Returns a pointer to first byte of memory
  · Returns NULL, if memory cannot be allocated

- Calloc: · void* calloc(size_t num, size_t sz)
  · Allocates a block of memory with size num x sz
  · Memory is filled with 0
- free: · void free(void*)
  · Releases memory at the pointer
  · Must point to the first byte malloced memory
- realloc: · void* realloc(void *ptr, size)
  · Changes the size of the pointer
  · Copies the data to a new location

Heap / Stack

- Stack: · variables without malloc allocation
- Heap: · allocated by malloc
- Memory leak: When code doesn't deallocate memory

Struct

- Like a class in Java, but without methods
- typedef struct quadrat {
  int length;
  int height;
  } quad
- quad quadrat1 <=> struct quadrat quadrat1
- Use "." to refer to fields in a struct
- Use "->" to refer to fields through a pointer to a struct ≡ (*p).x

---

Generic pointer

- void *generic_ptr
- Can be converted to any type of pointer without casting
- int *test = generic_ptr

Union

- Like a struct, but holds only one field at a time
- union u {
  int val;
  float fval;
  }

The C-Preprocessor (text substitution tool)

- Include source code: • #include <system_header.h>
  • #include "own_header.h"
  ↑ Everything with # is for the preprocessor
- Header-file: • file.h
  • contains only declarations

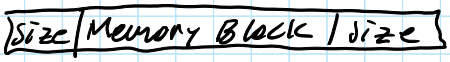
Jumps

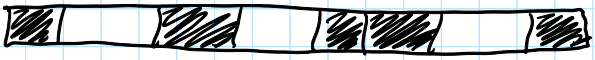
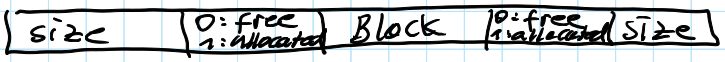
- setjmp(env)
  • Saves the current state in env
  • Returns 0
- longjmp(env, val)
  • Causes another return to the point saved by env
  • This return returns val
- Gut für Coroutines (siehe Grafik) oder Error-Handling

```c
void routineA()
{
    int r ;

    printf("(A1)\n");

    r = setjmp(bufferA);
    if (r == 0) routineB();

    printf("(A2) r=%d\n",r);

    r = setjmp(bufferA);
    if (r == 0) longjmp(bufferB, 20001);

    printf("(A3) r=%d\n",r);

    r = setjmp(bufferA);
    if (r == 0) longjmp(bufferB, 20002);

    printf("(A4) r=%d\n",r);
}
```

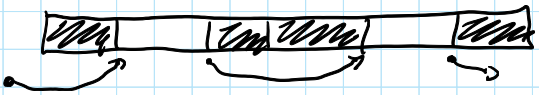
```c
void routineB()
{
    int r;

    printf("(B1)\n");

    r = setjmp(bufferB);
    if (r == 0) longjmp(bufferA, 10001);

    printf("(B2) r=%d\n", r);

    r = setjmp(bufferB);
    if (r == 0) longjmp(bufferA, 10002);

    printf("(B3) r=%d\n", r);

    r = setjmp(bufferB);
    if (r == 0) longjmp(bufferA, 10003);
}
```

Memory Allocation

- Used by malloc/calloc/realloc/free
- Current heap size: wird während Programmausführung immer grösser
- Peak memory utilization: · Am besten 100% (keine Verschwendung)
  · Wie viel Prozent des heaps ist
  · = $\frac{\text{Maximale Summe allocated Memory}}{\text{Heap size}}$
- Fragmentation: · Internal: 

  · External: 
- Implicit list: · 

  · Bidirectional Coalescing of free blocks
    - Start/end tags
    - Allows to traverse the list backwards
- Explicit list: · Using pointers to free blocks



- Segregated free list: · Different free lists to different block sizes
  · For larger sizes: one list for each power of two

Garbage Collector

- Common in functional languages, e.g. Java
- We can identify garbage, if there are no pointers to them
- Mark and Sweep garbage-collector with malloc/free
  1. Use extra mark-bit in the head of each block
  2. Start at root* and set mark bit on each reachable block
  3. Scan all blocks, and free them, if they are not marked

*e.g. global variables

Assembly

- Complex instruction set >< Reduced instruction set
- Transfer data between memory and registers
- Operand types: • Immediate: integer
  • Register: %eax
  • Memory: memory address given by register
    ↳ (%eax)
- Memory addressing: D(Rb, Ri, s) ⇒ Mem[Reg[Rb] + s * Reg[Ri] + D]
  (D: Constant, Rb: Base Register, Ri: Index Register, s: Scale)
- Condition Codes
  • Single bit registers: carry flag / zero flag / sign flag / overflow flag
  • Implicitly set by arithmetic-operations
  • Explicitly set by: - cmpX src2, src1 (Arithmetic)
    - testx src2, src1 (Bitwise)
- Conditional branch

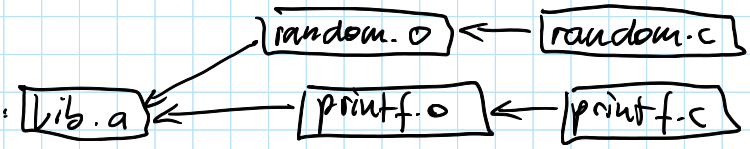
1. Setup
2. Test branch
3. Body 1 } if branch is taken
4. Body 2 } else branch is taken
5. Finish

Linker Symbols

- Global symbol: • Defined in module, that can be referenced by other
- External symbol: • Global symbol that are referenced by module m, but defined in module n
- Local symbol: • Symbol that is initialized and only used in the same module (static int temp = 5)
  • int temp is not a symbol

- Strong symbol: · procedures and initialized values
- Weak symbol: · Uninitialized global
  · compiler doesn't know, if it's an external variable, or an initialized one
- Only one strong symbol
- One strong symbol + several weak symbols } with the same name
- several weak symbols, pick random one
- Use prefix extern, if you "import" the variable

Libraries



- Static libraries:
- Shared/dynamic libraries: · object files, that are loaded into application at run-/compile-time

---

Executable and linkable format

- ELF
- Anything that ends in .o is an ELF-file
- .text section: Code
- .data section: initialized global variables
- .bss: · Bullshit section
  · Uninitialized global variables

Floating Point

- $(-1)^S \cdot M \cdot 2^E$ ≙ | s | exp | frac |
  - S: sign, M: Fraction, E: exp
- Single precision: 32 Bit: float
- Double precision: 64 Bits: double
- Casting: floating point → int: rounding to zero: 4,56 → 4
- Normalized values (good for big values)
  - Condition: exp ≠ 0...00 or exp ≠ 1...11
  - Exp: unsigned (coded as biased)
  - Bias: $2^{e-1} - 1$, where e is the number of exponent bits
  - E = Exp − Bias
  - Fraction $M = 1.xxxx\ldots x_2$
    - Minimal: M = 000....0 ≙ M = 1.0
    - Maximal: M = 111...1 ≙ M = 2.0
- Denormalized values (good for small values near zero)
  - Condition: exp = 00....0
  - Exponent value: E = −Bias + 1
  - exp = 00...0, frac = 000....0: represents value 0
  - exp = 00....0, frac ≠ 000....0: Number very close to 0.0
  - Fraction $M = 0.xxx\ldots x_2$
- Special values
  - Condition: exp = 11....1
  - frac = 000....0: ±∞
  - frac ≠ 000....0: not a number (=NaN)
- Rounding: Nearest even: if it's in the middle, round to even
  - ↳ 1,50 → 2
    2,50 → 2

Denormalized numbers	0 0000 000	-6	0	
	0 0000 001	-6	1/8*1/64 = 1/512	closest to zero
	0 0000 010	-6	2/8*1/64 = 2/512	
	…			
	0 0000 110	-6	6/8*1/64 = 6/512	
	0 0000 111	-6	7/8*1/64 = 7/512	largest denorm
Normalized numbers	0 0001 000	-6	8/8*1/64 = 8/512	smallest norm
	0 0001 001	-6	9/8*1/64 = 9/512	
	…			
	0 0110 110	-1	14/8*1/2 = 14/16	
	0 0110 111	-1	15/8*1/2 = 15/16	closest to 1 below
	0 0111 000	0	8/8*1 = 1	
	0 0111 001	0	9/8*1 = 9/8	closest to 1 above
	0 0111 010	0	10/8*1 = 10/8	
	…			
	0 1110 110	7	14/8*128 = 224	
	0 1110 111	7	15/8*128 = 240	largest norm
	0 1111 000	n/a	inf	

S exp frac E

Optimization

- Watch innermost loop
- Move code out of the loop
- Multiplication can be replaced with shifts
- Procedure calls are treatened as black-boxes

Architecture

- Superscalar processor: can execute multiple instructions in one cycle
- Cycles per element: · Execution time = CPE · n + overhead

Caches

- Miss rate: fraction of memory not found in cache
- Hit time: time to deliver a line in the cache to the CPU
- Miss penalty: additional time required because of a miss

↳ 99% hits is twice as good as 97% hits

· Cache hit time of 1 cycle >< miss penalty of 100 cycles

· 97% hits: 1 cycle + 0,03 · 100 cycles = 4 cycles

· 99% hits: 1 cycle + 0,01 · 100 cycles = 2 cycles

↳ That is, why miss rate is used, instead of hit-rate

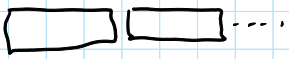
- Cache miss:
  - cold / compulsatory: first time you access a block
  - conflict miss: multiple blocks map to same location
  - capacity miss: cache to small for all active blocks
  - coherency miss: multiprocessor miss

- S: # sets
- E: # Blocks / Line per Set
- B: # Bytes per block

1. Set is located with set-index
2. Check if any block/line matches the tag
3. Data starting at the offset

- Direct-mapped-cache:
  - One block per set
  - No associativity

- Fully-associative-cache:
  - Only one set



- Private: only one core sees the cache
- shared: for multiple cores
- Inclusive: Everything in this cache is also in lower-level-caches
- Exclusive: Everything in this cache isn't in lower-level-caches

---

Cache writes

- Write hit
  - Writethrough cache:
    - Write immediatly back to memory
  - Writeback cache:
    - Wait to write back, until block gets replaced
    - Needs a dirty bit
- Write miss
  - Write-allocate cache:
    - Load into cache, update block in cache
  - No-write-allocate cache:
    - Writes immediatly to memory

Exceptions

- Each type of exception-event has a unique exception-number => index into exception-table
- Kernel-mode
  - Access to system state
  - Some new instructions / registers
  - For resolving the exception
- Synchronous exceptions
  - As a result of executing an instruction
  - Trap: intentional exception e.g. breakpoint
  - faults: recoverable e.g. page-fault
  - Aborts: unrecoverable e.g errors
- Asynchronous exceptions
  - Events of extern e.g. reset-button
  - Also called interrupts

Virtual memory

- Advantages: · Linear address space for each program
  · Isolates the programs
  · Efficient use of main-memory (=RAM)
- Page hit: 1. Process sends virtual address to MMU
  2. MMU fetches physical address from TLB
- Page fault: 1. Process sends virtual address to MMU
  2. MMU gets page from the disk

- Virtual: VPN | VPO
- TLB: TLB Tag | TLB Index
- Physical: PPN | PPO
- Cache: Cache Tag | Cache Index | Offset

Cache hit ← → Cache miss

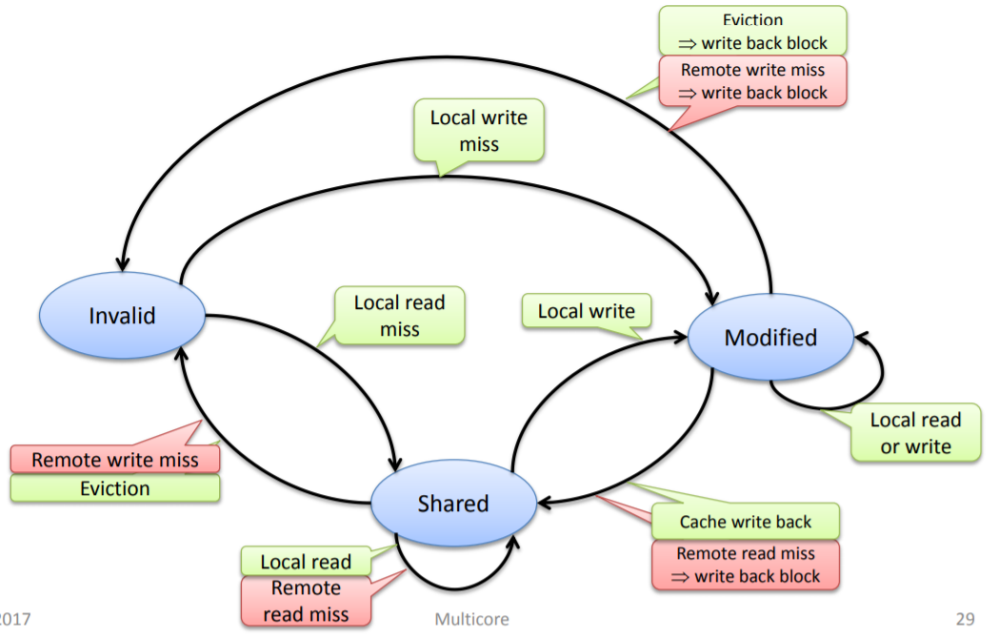
Cache addressing schemes

- Virtually-indexed, virtually tagged (VV) } Problem, same names for different physical locations
- Virtually-indexed, physically tagged (VP)
- Physically-indexed, physically tagged (PP)
- Physically-indexed, virtually tagged (PV)

Multiprocessing

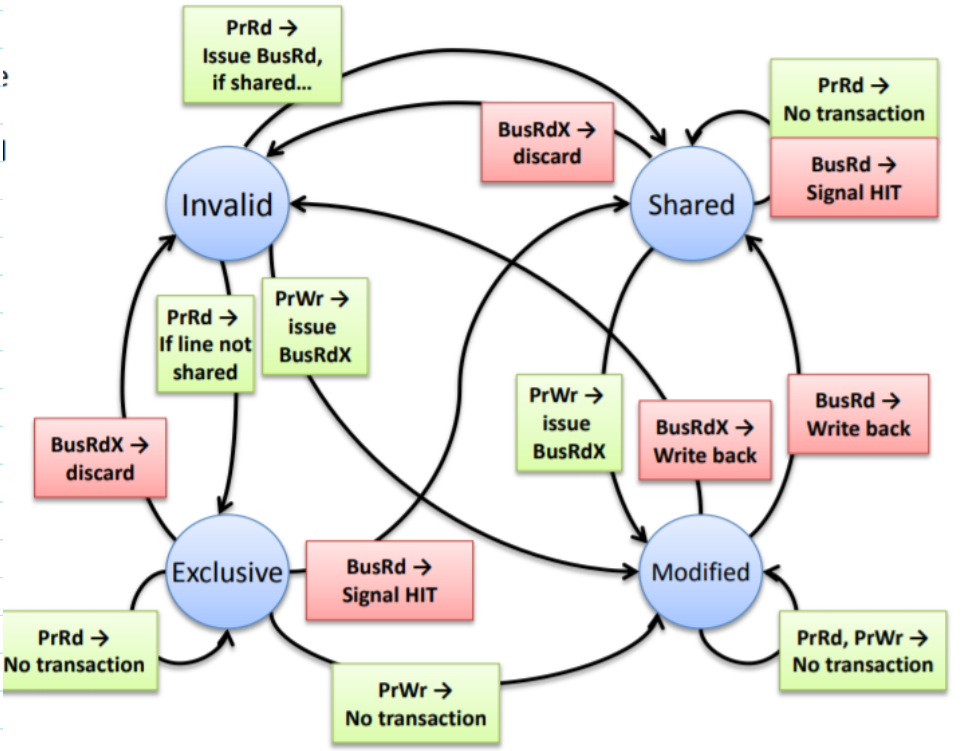
- Coherency: Values in all caches match each other
- Consistency: the order in which changes to the memory are seen by the CPUs

Cache coherence MSI



- Line can be dirty in only one cache
- Invalidate: removes element from cache, after writing back
- flush: write back

Cache Coherence MESI



Multicore 34

- Exclusive: Wenn datei niemand anders haben kann
- Advantage: If we are in exclusive and read/write, we must not broadcast that.

Devices

- Registers: · Obtain status info
  · Read input data
  · Write output data
- Device registers bypass the cache
- How does the CPU knows, when device is finished => Interrupts
- Direct memory access
  · Bypass CPU to transfer data directly between Device xRegister
  · Very simple processor => Only copies data
- Too many interrupts
  · Asynchronous buffering - ring
- Device initialization
  1. Wait for hardware to initialize
  2. Stop the device from doing anything
  3. Create buffer ring
  4. Write in register to start