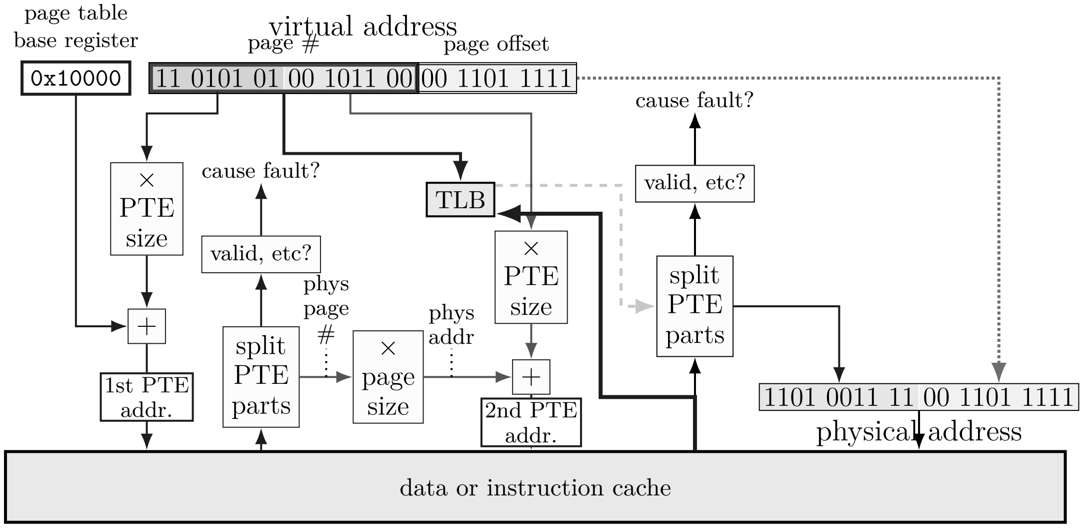
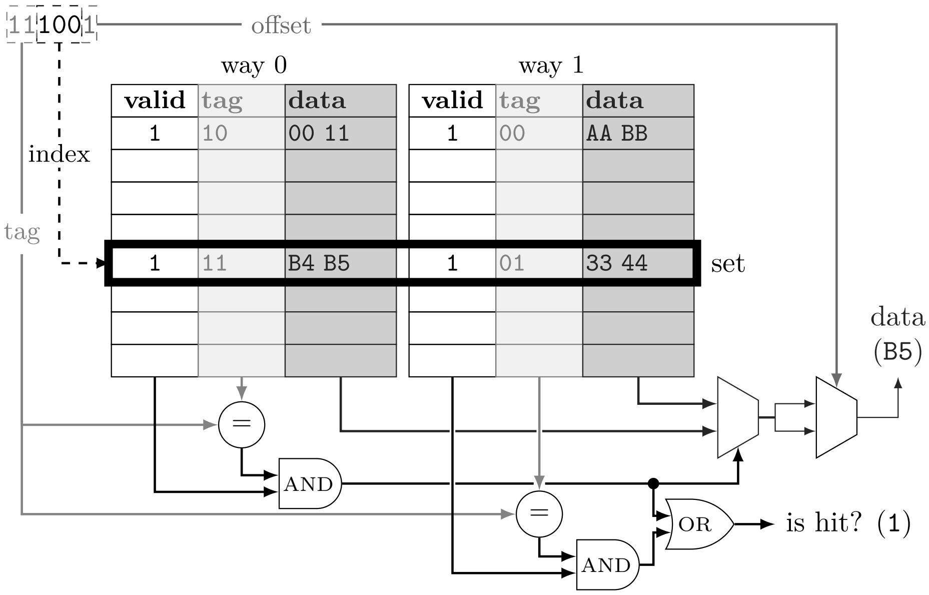
Spring 2024 3130 Final REFERENCE SHEET

1 page table lookup



2 cache organization

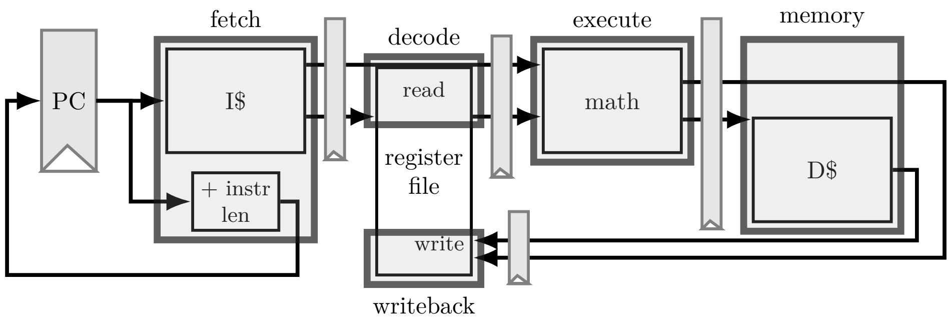


valid	tag	data	valid	tag	data
1	10	00 11	1	00	AA BB
1	11	B4 B5	1	01	33 44

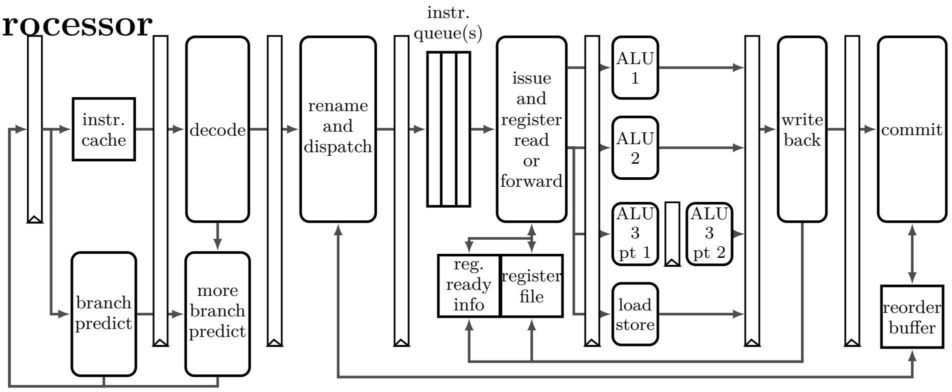
3 networking layers

application	HTTP, SSH, SMTP, ...	URLs, ...	...	application-defined meanings
transport	TCP, UDP, ...	port numbers, ...	segments, datagrams	reach correct program, reliablity/streams
network	IPv4, IPv6, ...	IP addresses, ...	packets	reach correct machine (across networks)
link	Ethernet, Wi-Fi, ...	MAC addresses, ...	frames	coordinate shared wire/radio
physical	...	...	...	encode bits for wire/radio

4 pipelined processor



5 OOO processor



6 selected POSIX functions

- give `lock` is a `pthread_mutex_t` and `cv` is `pthread_cond_t`
  - mutex lock/unlock: `pthread_mutex_lock(&lock)`; `pthread_mutex_unlock(&lock)`;
  - `pthread_cond_wait(&cv, &lock)` — unlock lock + wait on cv's queue; when woken up, relock lock and return; can be woken up early by 'spurious wakeup'
  - `pthread_cond_signal(&cv)` — wake up one waiting thread from cv's queue
  - `pthread_cond_broadcast(&cv)` — wake up all waiting threads from cv's queue
  - create new process copying current: `fork()` — return new pid in parent (old), 0 in child (new)
  - `pipe(fds)` — create a pipe, set fds[0] to the file descriptor for the read end, fds[1] for the write end
  - `write(fd, buffer, size)` write `size` bytes from `buffer` to the file descriptor `fd`
  - `read(fd, buffer, size)` read up to `size` bytes from `buffer` to the file descriptor `fd`, return total bytes read or 0 on end-of-file
  - `waitpid(pid, 0, NULL)` wait for the child process with ID `pid` to terminate
  - `kill(pid, signal_number)` — send signal `signal_number` to process `pid`
  - `sigaction(signal_number, &act_struct, NULL)` — configure signal handler for the specified signal based on the information in `act_struct`

7 assembly

- `OPq %r8, %r9`: perform OP (example: add) on `%r8` and `%r9`, put a resulting number (if any) in `%r9`
- `movq X, Y`: move 64-bit value from X to Y
- `%r8`, `%rax`, etc. — 64-bit register
- `(%r8)` — the value in memory at an address equal to the value of `%r8`